DAQ interface to Offline

Further Revised Proposal

12/30/98

M. W. Schulz, M. J. LeVine, and Brian Lasiuk

This version

There are two basic changes in this version:

- Addition of a new class with methods to access data by FEE and pin number to facilitate interface to pad monitors
- Removal of access to linearized ADC values for simulation purpose

The latter change was made because the ADC data are only available in 8-bit compressed form, so simulation should produce them in this form as well. The analysis code accessing simulated TPC data should use the same access methods as the code analyzing DAQ-produced data.

Introduction

This document specifies the innermost piece of the proposed data interface between DAQ data, provided either via network direct to Online or from events archived on mass storage. The proposed interface is meant to be virtually identical for DAQ data and for events provided by the STAR simulation tools.

The present draft of this document does not specify the uppermost layers of the interface [event, detector]. It specifies access to the TPC data at the sector level. The basic paradigm is that an entire event is read into memory, and one or more TPC sectors are instantiated as described below. The proposed interface is a compromise between required storage and transparency of access to the data, but it is intended to be useful in a variety of different programming styles.

The ADC data provided by DAQ are left in their 8-bit (compressed) form. The method **TpcSector::getSequences()** creates an array of structs for a single pad specified by (padrow, pad) which facilitates access to the zero-suppressed ADC data. Conversion to linearized (10-bit) form is done by accessing a lookup table with the 8-bit ADC value as index. This makes the source code look a bit awkward, but there should be little or no run-time penalty with modern compilers and sizable CPU caches. The motivation for this is to avoid the requirement for an additional 1MB of storage for the linearized ADC data for the sector.

A method **TpcFeeSector::getFeeSequences()** of the second new class creates an analogous array of structs, where the pad is specified by (fee,pin).

The basic struct Sequences contains a pointer to a compressed ADC value, the first on a pad. Its remaining element is a pointer to the array of structs **Sequence[ ]** which contain details of each timebin sequence of ADC values. The struct **Sequences** is used in both the **TpcSector** and the **TpcFeeSector** objects; only the method differs for specifying the pad.

Note that the time sequence returned by **getFeeSequences()** is not identical to the time sequence in the DAQ data format: in the DAQ format, a time sequence is limited to 32 bins. Physical sequences longer than 32 time bins are broken into pieces <32 timebins in the DAQ output. Occurrences of multiple segments will be reported as a single sequence by this interface, i.e., Humpty Dumpty will be glued together again.

TpcSector class

Unpacking is done by the **TPCSector** object. The Sector object's and PadRow struct's constructors allocate all of the storage required for the auxiliary structs. Deleting the Sector object releases this storage. This allows the application programmer to decide how many sectors' worth of data need to be resident at any given time.

A method **Sector::getPadList()** returns an array containing the numbers of those pads which contain valid ADC sequences.

TpcFeeSector class

The **TpcFeeSector** object does the unpacking. The FeeSector object's constructor allocates all of the storage required for the auxiliary structs. Deleting the Sector object releases this storage. This allows the application programmer to decide how many sectors' worth of data need to be resident at any given time.

The array **Sequences,** whose elements are returned one at a time by **TpcFeeSector::getFeeSequences(),** contains all the basic information necessary to navigate the ADC timebin sequences indexed by FEE, pin. Note that the entire array Sequences[ ][ ] is never visible outside the class, and, may in fact not be instantiated as a contiguous array in memory.

A method **Sector::getFeeList()** returns an array containing the numbers of those FEEs accessible in this sector of this event [which can vary when some FEEs are not present].

Definitions

```
struct Sequence
{
  unsigned short startTimeBin ; // the time of the sequence start
  unsigned short Length ;       // in units of time bins
  unsigned short Offset ;       // offset from the first hit of the pad
};

struct Sequences
{
unsigned char*  FirstAdc    ;   // pointer to the first hit
sequence* Seq ;                 // pointer to the sequence array
};

// The following struct is internal to the TpcDaqSector object
//(unpacker). It is described here to provide an idea of the storage
// required by the TpcDaqSector object.

struct PadRow
{
  unsigned char* nSeq ;
  Sequences**    Seq ;
  PadRow(int Pads) ;//argument:# pads this pad row
                    //allocates space needed for pad row
  ~PadRow() ;
  clear() ;       //cleans up storage used for Sequences, zeroes nSeq[];
  int nPads ;     // variable pad size
};
```

Interface

```
//base class for both TpcDaqSector and TpcSimSector classes:
```

class TpcSector

```
{
  public:
            // makes the data for sector accessible:
      virtual int getSector(int which, Event* event) = 0 ;
            // 0 == O.K.
            // -1 wrong sector number,
            // -2 wrong event
            // method to get access to the data:
      virtual int getSequences(int padRow,
            int pad, int *nSeq, Sequences** Seq) = 0 ;
            // 0 == O.K. -1 wrong Row, -2 wrong pad,
            // -3, -4 wrong third and fourth arg,
            //-1000 no call to getSector

      virtual int getPadList(int PadRow, unsigned char **PadList) = 0;
            // returns number of pads in array PadList[]
            // >0 return value: no sequences on this pad row
            // -1000: getSector call failed
            // PadList[] is an array of pad numbers which
            // contain one or more valid sequences

            // removes or resets internal aux structs:
      virtual void clear() = 0  ;
};
```

class TpcDaqSector: public TpcSector

```
{
public:
      int getSector(int which, DaqEvent* event);
      int getSequences(int padRow, int pad, int* nSeq, Sequences** Seq);
      int getPadList(int PadRow, unsigned char **PadList);
      Sector();
      ~Sector();
      clear() ;
private:
      PadRow* Row[45] ;
      int status ;
      // 0 not filled,
      // 1 filled O.K
      // >0 filled, with problems
};
```

class TpcSimSector: public TpcSector

```
{
// This will be defined by Brian L for the Simulation derived class.
}
```

class TpcFeeSector ;

```
{
public:
      int getFeeSector(int which, DaqEvent* event);
      int getFeeSequences(int Fee, int Pin, int* nSeq, Sequences** Seq);
      int getFeeList(unsigned char **FeeList);
      FeeSector();
      ~FeeSector();
      clear() ;
private:
      int status ;
      // 0 not filled,
      // 1 filled O.K
      // >0 filled, with problems
};
```

Header file: TpcDataInterface.hh

```
#ifndef TPCDATAINTERFACE_HH
#define TPCDATAINTERFACE_HH
// Event belongs one level higher. It is included here only for
// completeness
class Event
{
public:
  virtual int getEventNumber() = 0 ;
  virtual int getRunNumber() = 0  ;
};

struct Sequence
{
  unsigned short startTimeBin ; // the time of the sequence start
  unsigned short Length ;       // in units of time bins
  unsigned short Offset ;       // offset from the first hit of the pad
};
struct Sequences
{
  union
  {
    unsigned short* FirstLinAdc ;
    unsigned char*  FirstAdc    ; // pointer to the first hit
  };
  Sequence* Seq ;
};
class TpcSector
{
public:
  virtual int getSector(int which, Event* event) = 0 ;
            // 0 O.K.,-1 wrong sector number, -2 wrong event
  virtual int getSequences(int padRow, int pad,
            int *nSeq, Sequences** Seq) = 0 ;
            // 0  O.K. -1 wrong Row, -2 wrong pad,
            // -3, -4 wrong third and fourth arg,
            //-1000 no call or failed call to getSector
  virtual int getPadList(int padrow, unsigned char** Padlist) = 0 ;
            // return value >= 0 := number if pads in array PadList
            // -1                := wrong padrow
            // -1000             := no successful call to getSector
  virtual void clear() = 0 ;
            // removes or resets internal aux structs:
};

unsigned

class TpcFeeSector
{
```

```
public:
  virtual int getFeeSector(int which, Event* event) = 0 ;
            // 0 O.K.,-1 wrong sector number, -2 wrong event
  virtual int getFeeSequences(int Fee, int Pin,
            int *nSeq, Sequences** Seq) = 0 ;
            // 0  O.K. -1 wrong Row, -2 wrong pad,
            // -3, -4 wrong third and fourth arg,
            //-1000 no call or failed call to getFeeSector
  virtual int getFeeList(unsigned char** Padlist) = 0 ;
            // return value >= 0 := number if pads in array FeeList
            // -1000             := no successful call to getFeeSector
  virtual void clear() = 0 ;
            // removes or resets internal aux structs:
};

unsigned
#endif // TPCDATAINTERFACE_HH
```

Example application - access by padrow, pad

This first example is correct, but suffers from being written explicitly for the **TpcDaqSector** derived class, thus not reusable in the simulation environment.

The loop would look like:

```
while(daqReader.readEvent(&event) > 0)
      // this is outside the interface defined here
{
  if(!daqReader.Tpc) continue ; // No TPC
  for(sector = 1 ; sector <= 24 ; sector++) // sector loop
    {
     int retval=TpcDaqSector.getSector(sector, event);
     if ((retval)
     {
        printf("this sector has a problem: %d\n",retval);
        exit(-1);
     }
     for(padrow = 1 ; padrow <= 45 ; padrow++)
      {
        npads = TpcDaqSector.getPadList(padrow,padlist);
        for(ipad=0, pad=padlist[0] ; ipad<npads ; pad=padlist[ipad++])
          {
            Sequences* TpcSequence ;
            iret =
            TpcDaqSector.getSequences(padrow,pad,&nSeq,&TpcSequence);
            if(iret < 0)
            {
                  printf("error message.. :-(   ");
                  return(iret) ;
            }
            if(!nSeq) continue ;  // no longer required!
            for(iseq = 0 ; iseq < nSeq ; iseq++)
            {
              // LinArray is a 256 element array of unsigned shorts
              // for linearizing the 8 bit adc values

              unsigned char *adc = TpcSequence->FirstAdc +
                                    TpcSequence->Seq[iseq].Offset;
              for(int t = 0; t < TpcSequence->Seq[iseq].Length ; t++)
              {
                  printf("Time %d Adc %d \n",
                        TpcSequence->Seq[iseq].startTimeBin + t,
                        LinArray[adc[t]]);

              } // end of loop over adc values
            } // end of loop over sequences
          } //  end of loop over pads
      } // end of loop over pad rows
      TpcDaqSector.clear();
    } // end of loop over sectors
} //end of loop over events
// ..
```

Better example - access by padrow, pad

The following application is an illustration of user code which would be usable in both the simulation and reconstruction environments, since in the demo loop it refers only to the base class **TpcSector**. In the outermost part of the code [see *main( ),* below], there must of course be a reference to the derived class that will be used.

```
// This code provides an example that passes a compile step and
// illustrates the interface as proposed. The higher level of the
// interface, the Event and Reader level are here because there
// has to be an upper level to make it "look" complete. These levels
// are presently completely undefined and have not even started to be
// discussed. The code included in this example declaring part of
// these upper levels is not a starting point for a discussion.


#include "TpcDataInterface.hh"
#include "TpcDaqDataInterface.hh"
#include "DummyClassesForDemo.hh"
#include <stdio.h>
#include <string.h>

// demo loop using the interface only

int demoLoop(Reader& reader, Event& event, TpcSector& tpcSector)
{
  int sector, padrow, npads, pad, iret   ;
  unsigned char* padlist ;
  unsigned short* LinArray = TpcLinArray ;

  while(reader.readEvent(&event) > 0)
            // this is outside the interface defined here
    {
      if(!reader.testDetector("TPC")) continue ; // No TPC
      for(int sector = 1 ; sector <= 24 ; sector++) // sector loop
      {
        int retval = tpcSector.getSector(sector, &event);
        if (retval) {
            printf("this sector has a problem: %d\n",retval);
            exit(-1);
        }
        for(padrow = 1 ; padrow <= 45 ; padrow++) {
            npads = tpcSector.getPadList(padrow,&padlist);
            for(int ipad=0, pad=padlist[0] ; ipad<npads ;
                              pad=padlist[ipad++]) {
```

```
              Sequences* TpcSequence ;
              int nSeq ;
              iret =
                tpcSector.getSequences( padrow,pad,&nSeq,&TpcSequence);
              if(iret < 0) {
                  printf("error message.. :-( "); return(iret) ;
              }
              if(!nSeq) continue ; // no longer required!
              for( int iseq = 0 ; iseq < nSeq ; iseq++) {
                  // LinArray is a 256 element array of unsigned shorts
                  // for linearizing the 8 bit adc values
                  unsigned char *adc =
                  TpcSequence->FirstAdc + TpcSequence->Seq[iseq].Offset;
                  for(int t=0; t < TpcSequence->Seq[iseq].Length ; t++)
                  printf("Time %d Adc %d \n",
                        TpcSequence->Seq[iseq].startTimeBin + t,
                        LinArray[adc[t]]);
              } // end of loop over sequences
            } // end of loop over pads
          } // end of loop over pad rows
          tpcSector.clear() ;
      } // end of loop over sectors
    } //end of loop over events
  return(0) ;
};
```

main(), illustrating sim and daq data reading

```
main(int argc , char** argv)
{
// setup the linarray (simplest case)
  for(int i = 0 ; i < 256 ; i++) { TpcLinArray[i] = i ; } ;

// are we simulating or analyzing DAQ data?

  if(!strncmp(argv[1],"SIM", strlen("SIM"))) //SIM case
    {
      SimEvent sEvent ;
      SimReader sReader;
      TpcSimSector sSector ;
      int iret =  demoLoop(sReader,sEvent,sSector) ;
      return (iret) ;
    }
  else // DAQ
    {
      DaqEvent dEvent ;
      DaqReader dReader;
      TpcDaqSector dSector ;
      int iret =  demoLoop(dReader,dEvent,dSector) ;
      return (iret);
    }
}; // end of main
```

Better example - access by Fee, pin

```
// This code provides an example that passes a compile step and
// illustrates the interface as proposed. The higher level of the
// interface, the Event and Reader level are here because there
// has to be an upper level to make it "look" complete. These levels
// are presently completely undefined and have not even started to be
// discussed. The code included in this example declaring part of
// these upper levels is not a starting point for a discussion.


#include "TpcDataInterface.hh"
#include "TpcDaqDataInterface.hh"
#include "DummyClassesForDemo.hh"
#include <stdio.h>
#include <string.h>

// demo loop using the interface only

int demoLoop(Reader& reader, Event& event, TpcFeeSector& tpcFeeSector)
{
  int sector, fee, pin, iret   ;
  unsigned char* feelist ;
  unsigned short* LinArray = TpcLinArray ;

  while(reader.readEvent(&event) > 0)
            // this is outside the interface defined here
    {
      if(!reader.testDetector("TPC")) continue ; // No TPC
      for(int sector = 1; sector <= 24; sector++) // sector loop
      {
        int retval = tpcFeeSector.getFeeSector(sector, &event);
        if (retval) {
            printf("this sector has a problem: %d\n",retval);
            exit(-1);
        }
        nfee = tpcSector.getFeeList(&feelist);
        for(int ifee = 0, fee=feelist[0]; ifee <= nfee; fee=feelist[ifee++]) {
          for(pin=0; pin<16; pin++) {
            Sequences* TpcSequence ;
            int nSeq ;
            iret =  tpcFeeSector.getFeeSequences(fee,pin,&nSeq,&TpcSequence);
              if(iret < 0) {
                  printf("error message.. :-( "); return(iret) ;
              }
              if(!nSeq) continue ; // some pins not connected!!
              for( int iseq = 0 ; iseq < nSeq ; iseq++) {
                  // LinArray is a 256 element array of unsigned shorts
                  // for linearizing the 8 bit adc values
                  unsigned char *adc =
                  TpcSequence->FirstAdc + TpcSequence->Seq[iseq].Offset;
                  for(int t=0; t < TpcSequence->Seq[iseq].Length ; t++)
                  printf("Time %d Adc %d \n",
                        TpcSequence->Seq[iseq].startTimeBin + t,
                        LinArray[adc[t]]);
              } // end of loop over sequences
            } // end of loop over pins
          } // end of loop over FEEs
          tpcFeeSector.clear() ;
      } // end of loop over sectors
    } //end of loop over events
  return(0) ;
};
```

main(), illustrating daq FEE data reading

```
main(int argc , char** argv)
{
// setup the linarray (simplest case)
  for(int i = 0 ; i < 256 ; i++) { TpcLinArray[i] = i ; } ;

   DaqEvent dEvent ;
   DaqReader dReader;
   TpcFeeSector dSector ;
   int iret =  demoLoop(dReader,dEvent,dSector) ;
   return (iret);
}; // end of main
```