DRAFT February 16, 1998

From Data Streams to Data Structures and Beyond
An Object-Oriented View of the CDMS Data Processing
Steve Eichblatt, Fermilab

1 Introduction

A set of objects have been written to transform the CDMS data *stream* into a data *structure*. This is analogous to transforming a pile of bricks, wood and furniture into a house...the raw ingredients are the same, but concepts like "bedroom" and "kitchen" make the use of a structure more convenient. In the CDMS data case, the "detector" is an example of a useful concept that comes with a data structure.

The object which transform the stream to the structure are complex, but easy to use in a c++ program. Once *instantiated*, they can be used to manipulate the structures directly. This leads to small, simple programs to perform analysis on the data. Two such programs will be presented here: one which reduces the data to a set of measured quantities (*à la* darkPipe), and another to display an event graphically on the screen.

It is also possible, although there are a few technical hurdles to cross, to save the data as a structure. Of course this is trickery, since a file is a stream, but an object-oriented data base can make it look like a structure. This is clearly a future goal.

2 Design

2.1 The Data Stream

Figure **??** shows the proposed layout of a data stream element. In this figure an *Event* refers to any type of event, be it data or a reading from the DMM, or a change in the online electronics...any self-contained packet of information about the data being taken.

A *Logical Record* is a unit of information within the event. In the case of a data event, it could be a BLIP Q_{inner} trace. Or it could be the timestamp record, or the online RQ's. In the case of a DMM type event, a logical record could be a particular voltage reading.

2.2 The Data Structure

```
// ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ Event
class Event{     // an Event Base class.
enum tracetype { CHARGE , PHONON };
protected:
  int nevent;
  int size;
  int    ntraces;
  char filename[100];
  char *buffer_origin;
  void fillScaler(char **buffer);  // fill scaler info
  void fillTraces(char **buffer);  // fill trace info
  void fillVeto  (char **buffer);  // fill veto info
  Geometry *AllDetectors;
  Geometry DetList;
public:
```

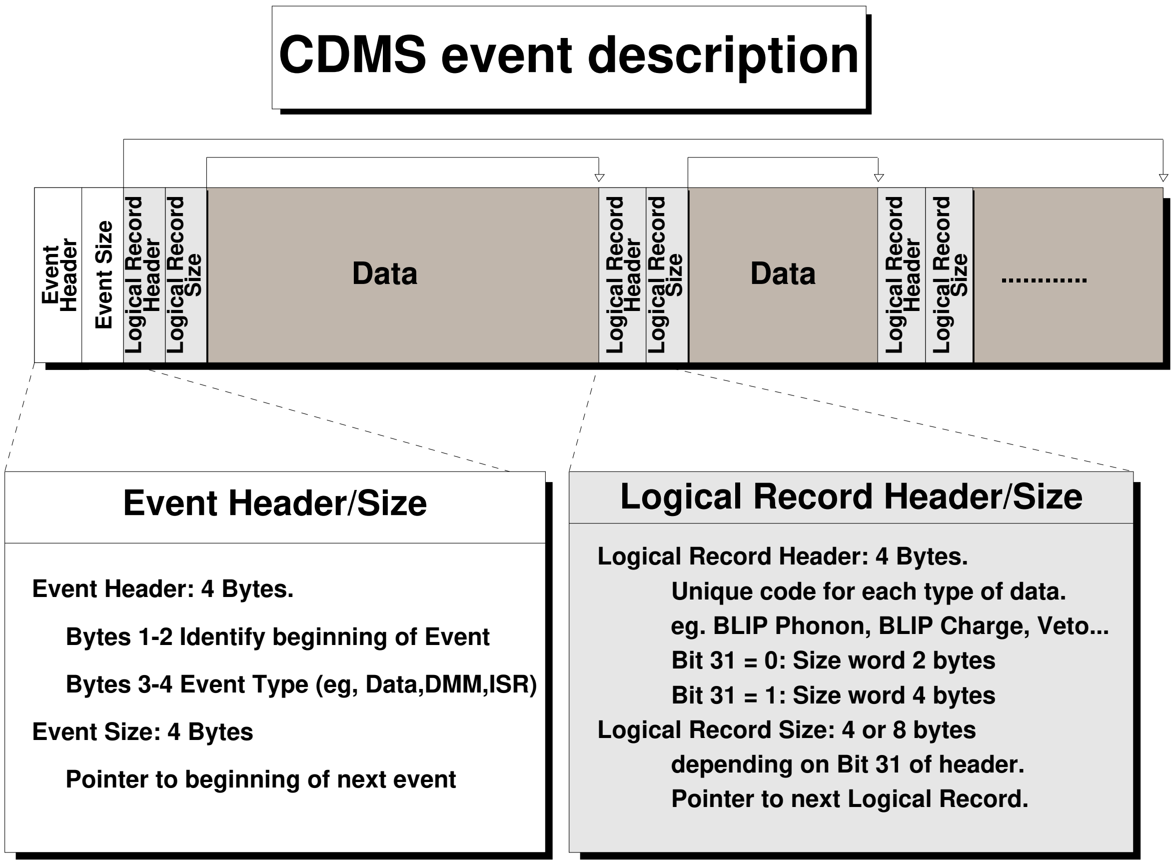

Figure 1: *A drawing of a proposed CDMS data stream element.*

```
  Scaler scaler;
  Veto   veto;
  Trace *trace;
  inline int nTraces(){return ntraces;}
  int read(netConnection *netC,int start);// return number of bytes read
  int read(FILE *fp);    // return number of bytes read
  int load(char **buffer);          // load event sturcture from byte array.
  void makeRQ();
  virtual int write(){;}            // return number of bytes written.
  virtual int dump(){;}             // return number of bytes dumped.
  virtual void print(bool printTrace);
  virtual void print(int thresh=-1);
  virtual void display(){;}
  int whichFired(void);             // which detector triggered.
  void Clear();                     // destroy objects created;
  Geometry& GetGeom() {Geometry& g=DetList; return g;}
  void SetGeom(Geometry *AllDet)   {AllDetectors = AllDet;}
// ----------------------------------------------------------- constructor
```

```
  Event(){
      size = 0; nevent =0; ntraces=0; AllDetectors = 0;
  }
// ----------------------------------------------------------- destructor
  ~Event(){
    delete [] buffer_origin; buffer_origin=NULL;
    delete [] trace;
  }
};
```

3 Applications

3.1 Data Reduction (darkPipe)

3.2 Event Display

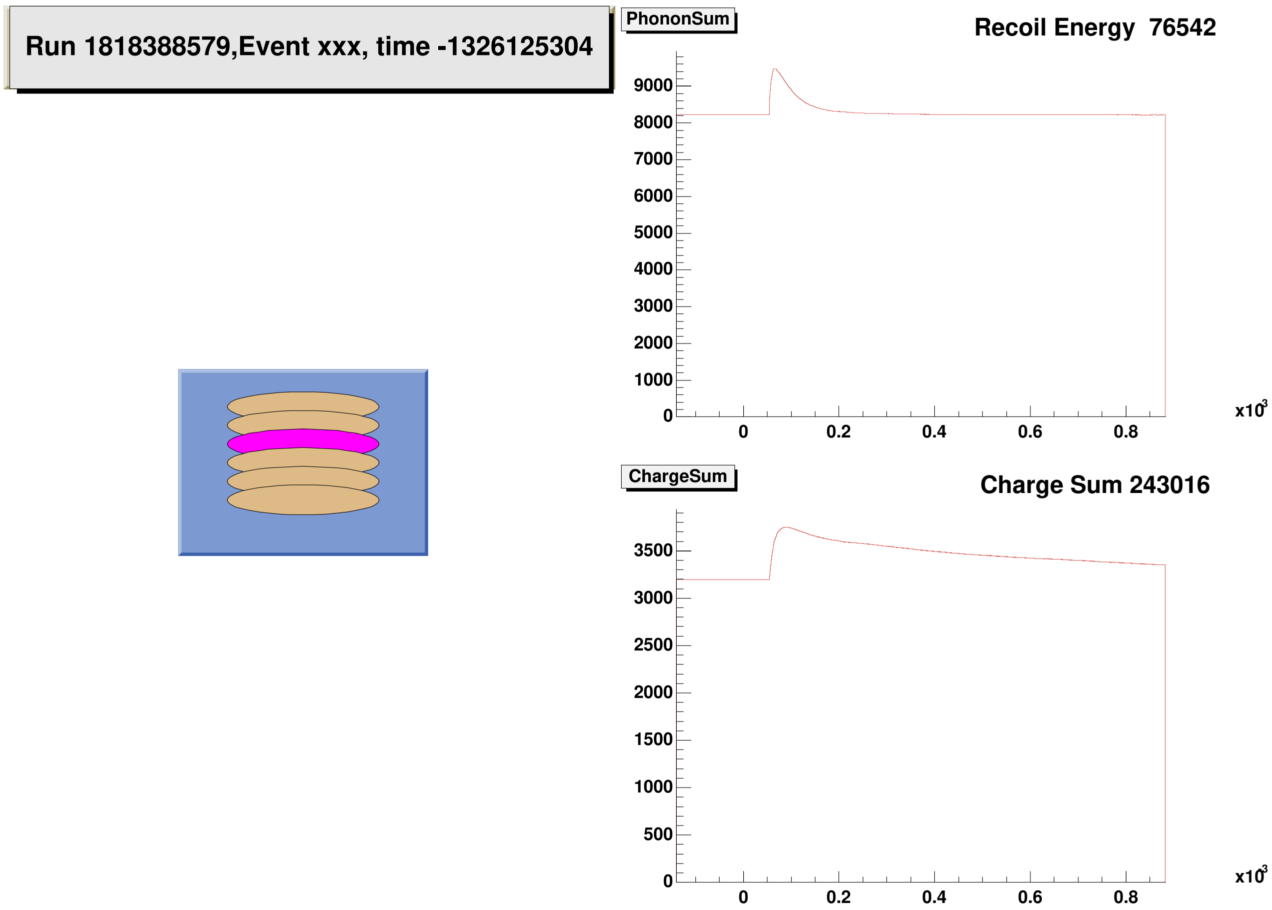

Figure 2: *An Event drawn by the event display program*

4 Plans

5 Conclusion