High Performance Output

G. Gavalian (Jlab)

Hipo

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Design Concepts

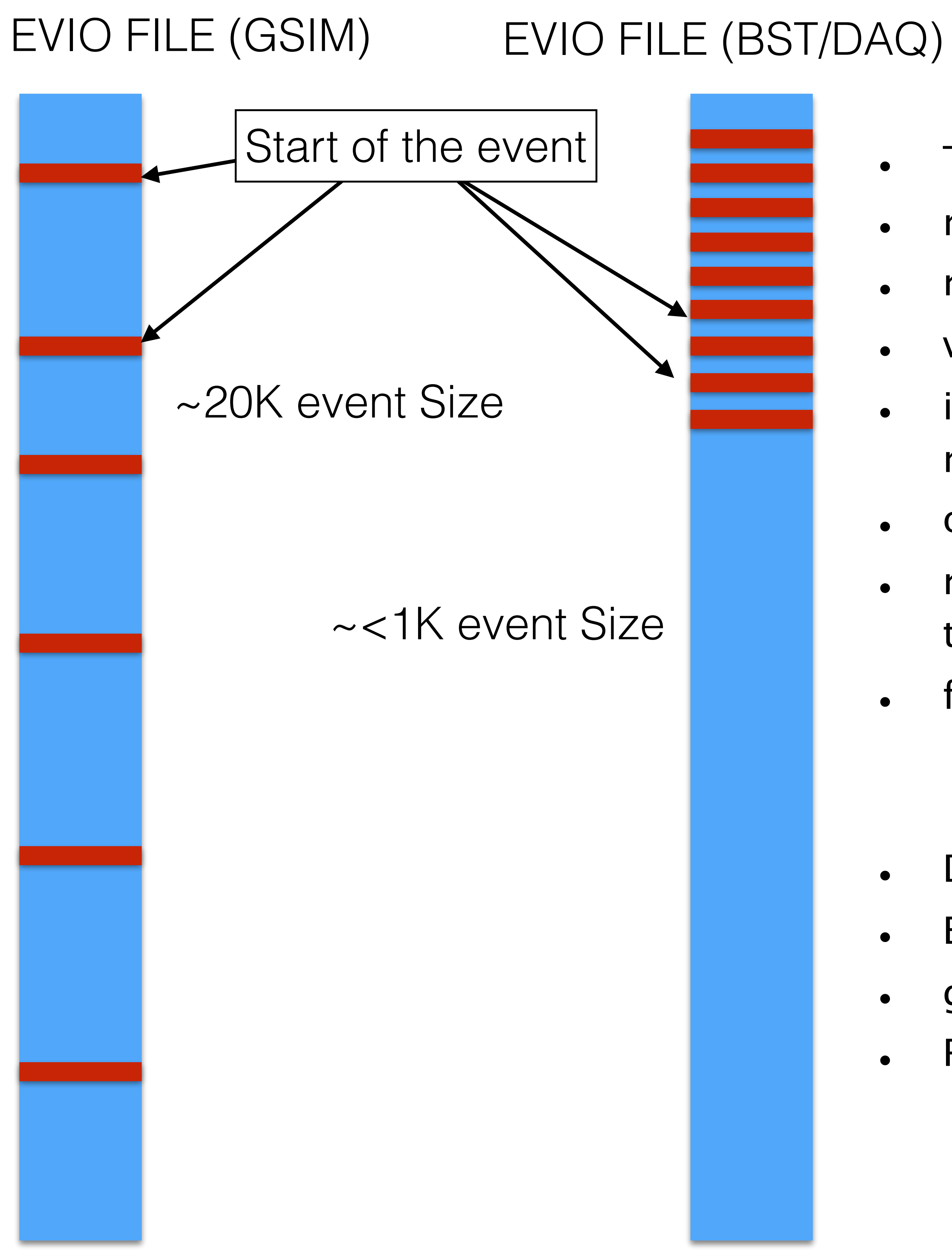
- **Requirements for Data File Format:**
  - Abstract away from data format used inside, easily can change implementations
  - Provide indexing of the data, without reading much of the file
  - Random access to any record in the file
  - Easy append, merge and split of the file
  - Provide ability to compress certain parts of the stream
  - Easy corruption recovery and re-indexing
- **What Options are available:**
  - **ROOT provides structured data storage:**
    - Not very good for data transfer over network in CLARA environment
    - Tied to the C++ classes, can not be used outside of ROOT
  - **EVIO event by event data buffer for DAQ:**
    - No functionality for easy append and drop banks
    - Can not write large files
    - Does index the data well for random access
    - no corruption recovery
    - no compression algorithm

Prehistory

- **What is out there:**
  - ROOT – is not used everywhere as many people claim
  - CERN uses variety of data formats (HEPEVT, HEPSIM, proMC, ROOT,...)
  - HDF – used mainly by astrophysics community
  - NetCDF – astrophysics community aimed to large object storage
- **New Trends:**
  - self descriptive data formats with object model.
  - Thrift (apache) – binary data packing framework optimized for data exchange.
  - protocol Buffers (google) – message exchange framework with self documented data (CERN started using it).
  - Avro – is a remote procedure call and data serialization framework developed within Apache's Hadoop project
- **Need to distinguish:**
  - all above mentioned protocols are for data encoding.
  - the do not provide solutions for data storing and distribution
  - need to separate data encoding and serialization framework from data storage

- **Example : Videos on the computer are all encoded (serialized) with H.261 codec (evio analog)**
- **Video files have different container formats: AVI, XVID, DVIX, MKV, MP4v and so on..**

EVIO

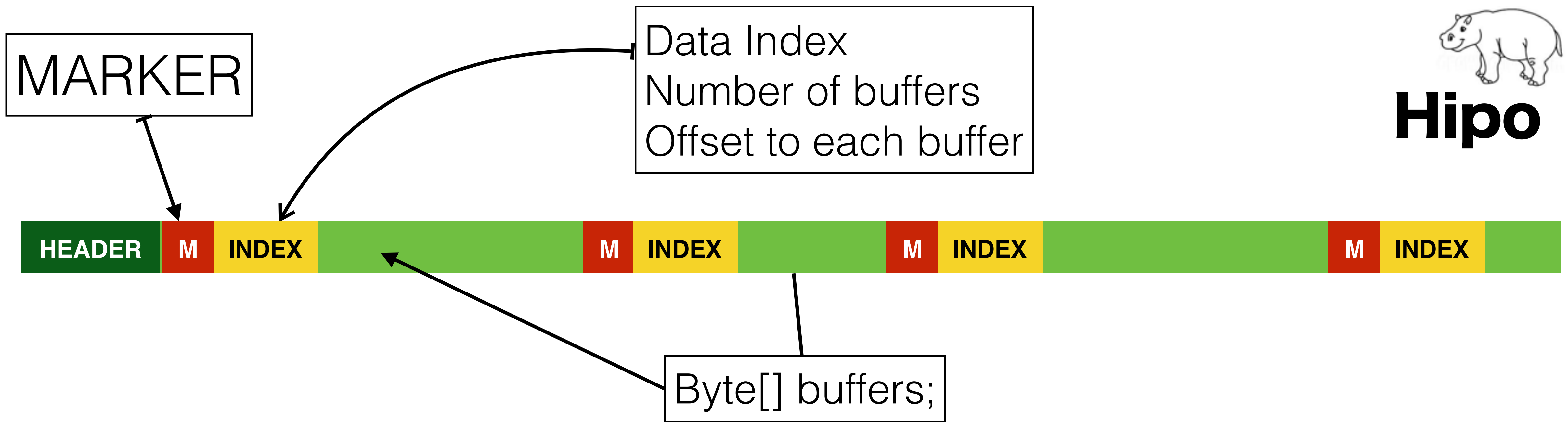


- The File design was just appending events.
- requires scanning entire file to index it.
- no flexibility to remove banks within the event.
- very large memory footprint
- individual parts are not accessible without reading the file
- old fashion sequential read implementation
- no API to group events together to pass over the network.
- file limit 2GB

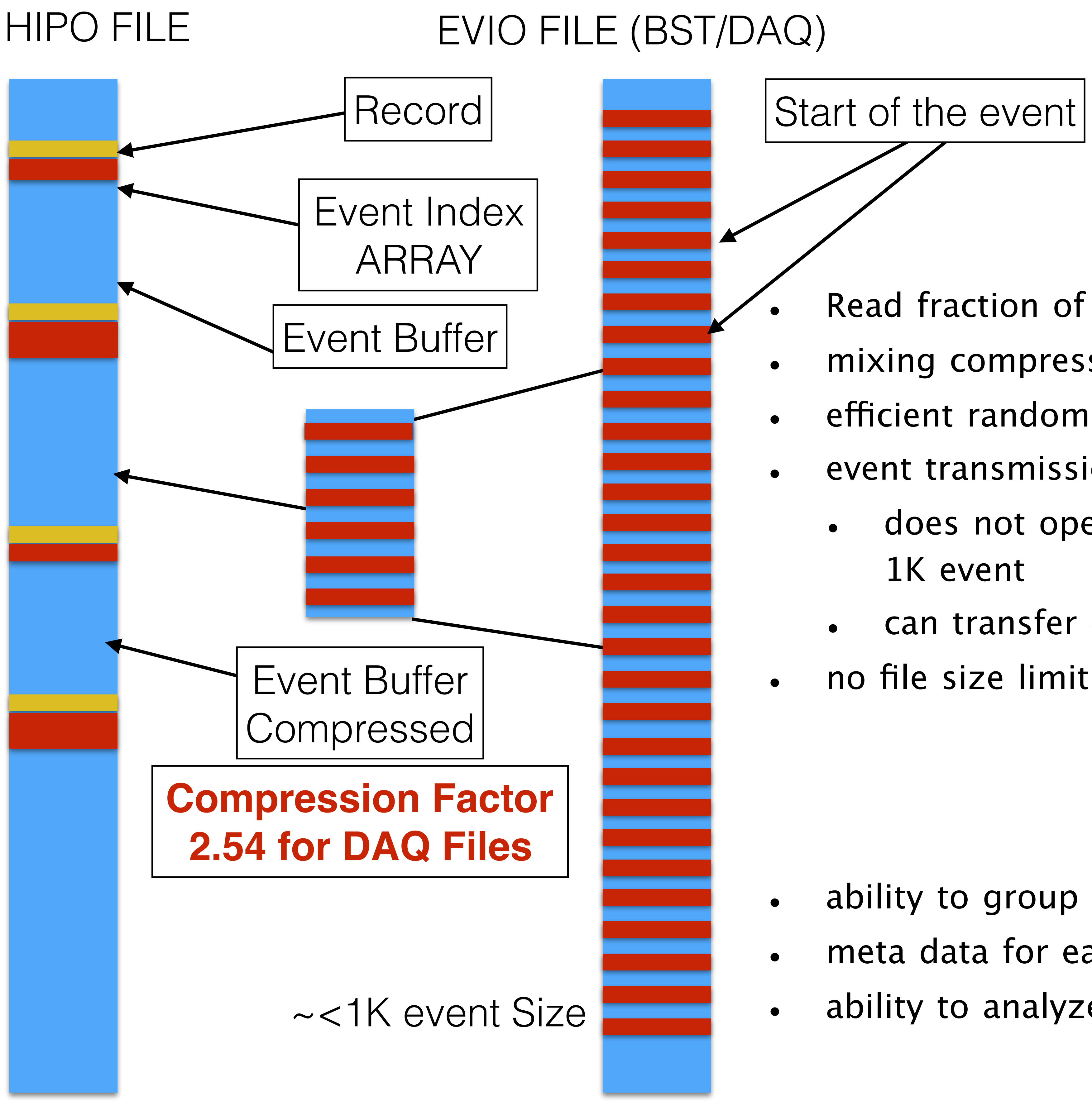
- DST files are going to have small event size
- Easier to work with them as a group of events
- group containing SCALER info
- Run conditions

Stream Structure

- Marker is a predefined String written each time write operation is performed.
- Index is int[] array listing relative offsets of each buffer in the stream.
- Data section is continuous Byte[] array, it will be decided according to indices.
- Data buffers can be compressed, index is only valid for non compressed array.
- Two modes of indexing are implemented:
    - Full Indexing – only MARKERS and INDEX arrays are read
    - Partial Indexing – only MARKERS are read
- Efficient first pass over the properties of the file.



EVIO



- Read fraction of file to get the index.
- mixing compressed and uncompressed data
- efficient random access through network (HTTP)
- event transmission through network by GROUP
  - does not open and close sockets for every 1K event
  - can transfer 4MB (200K events at once)
- no file size limit.

- ability to group events by Run conditions
- meta data for each group (RECORD) of events
- ability to analyze several files in parallel

Users Experience

MK

Here are some of my numbers.
2 evio files of 1.9GB (3.7GB total) to 1 hipo file of 1.9GB
Program took 112.402 seconds to read 2 evio files
Program took 53.053 seconds to read hipo file

9 evio files of 1.9GB (17GB total) to 1 hipo file of 8.4GB
Program took 346.289 seconds to read 9 evio files
Program took 177.061 seconds to read hipo file

RD

11 files, 850MB each (9 GB) , I obtained a 3.8 GB file.
The time needed to run the analysis on the hipo file is 3 times faster.

CS

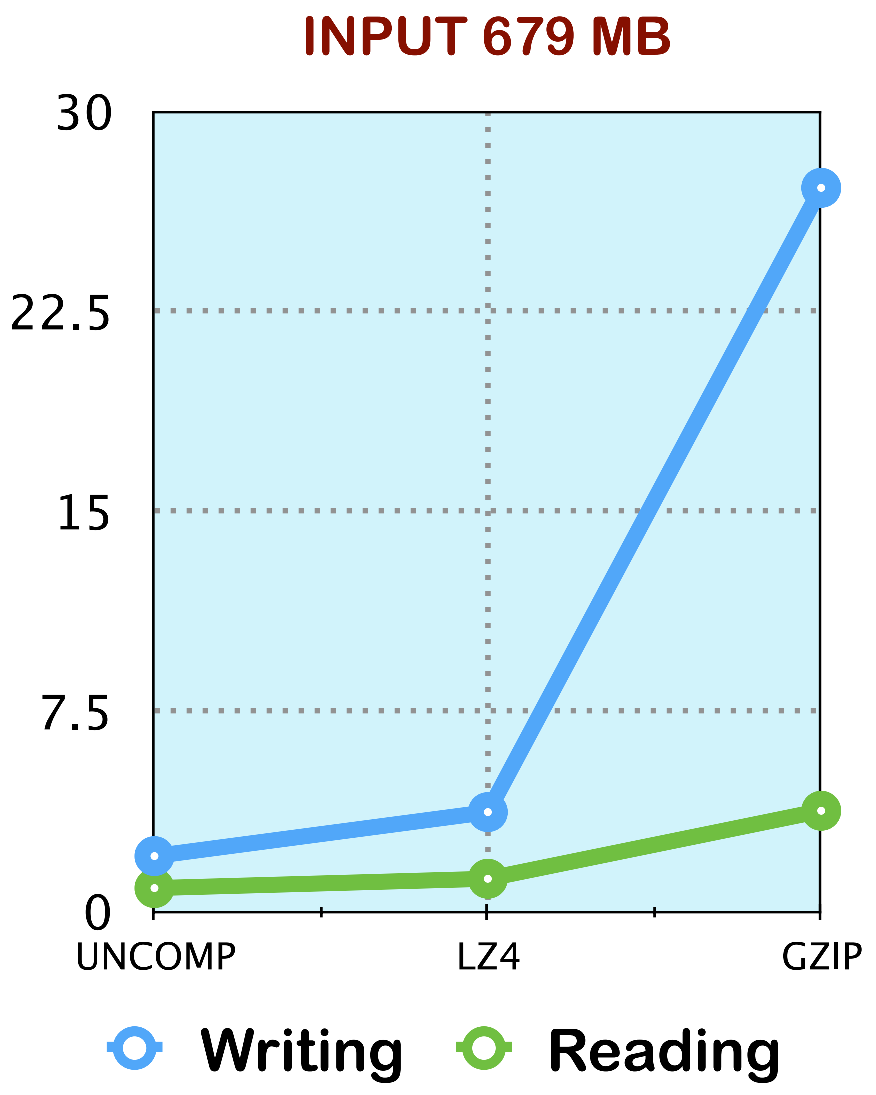
For 500K cosmic muons in ECAL,PCAL,FTOF dgtz banks I get reduction from 509 MB to 129 MB. There is no startup delay in reading the headers. Actual event processing time in ECMON reduced from 72 s to 63 s (2MB/s) with minor GUI activity. Looks promising!

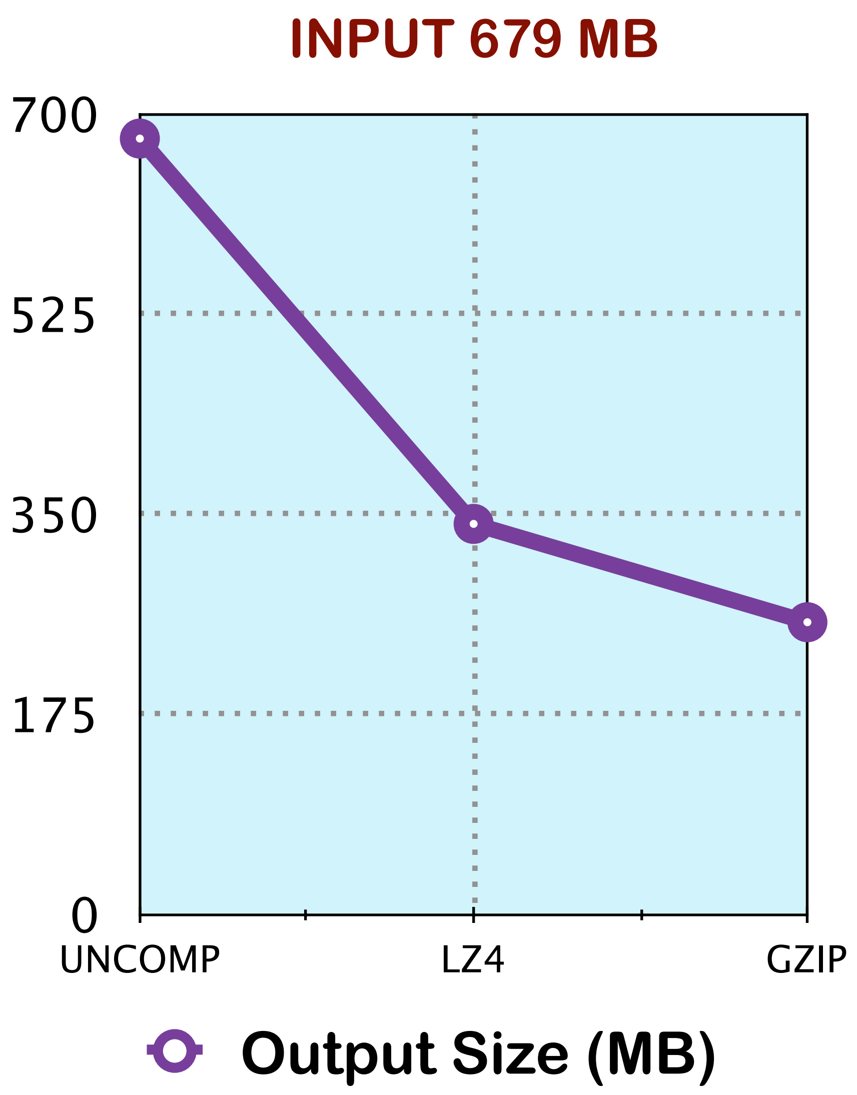
VZ

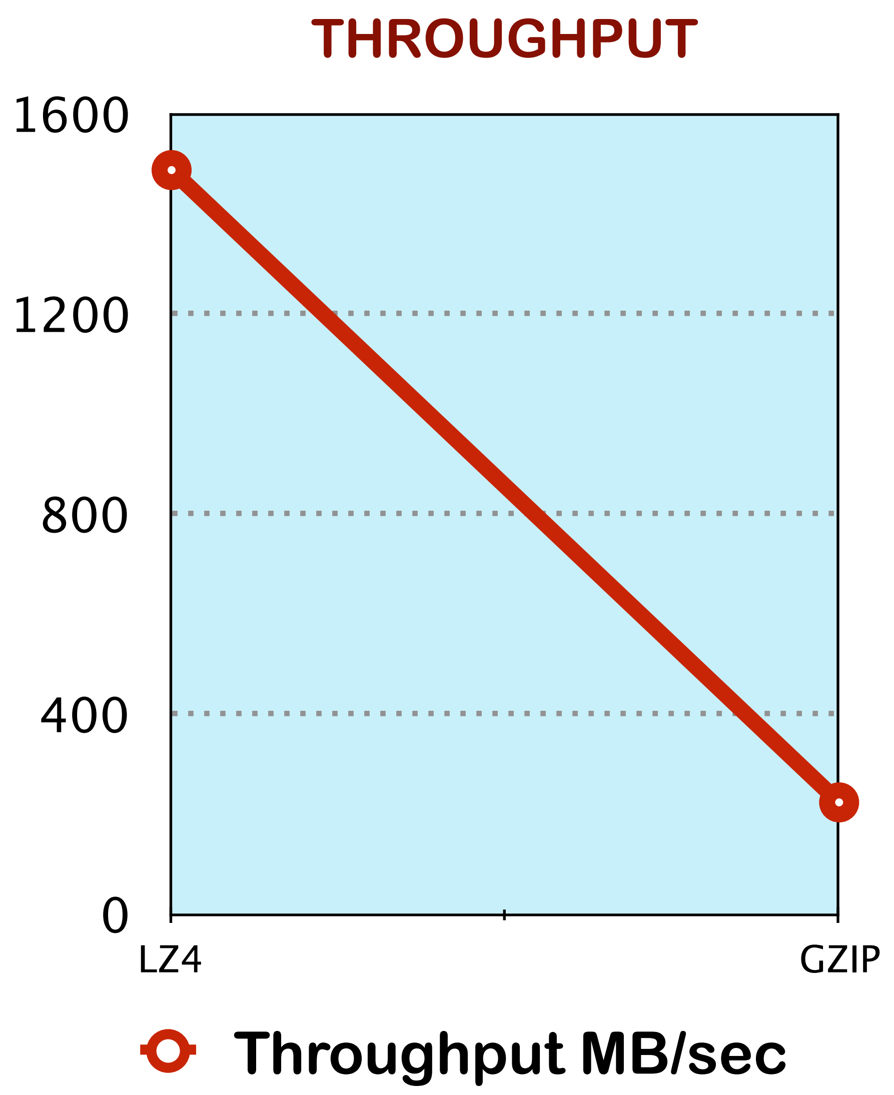
```
NRECORDS =        882, BYTES =  2470.76 Mb, WTIME =   4.744 sec, CTIME = 229.900 sec,
Files used in making the hipo file:
1.4G      /Users/ziegler/Desktop/Release-1-tracking-results/multi_forward_0002.0.evio
1.4G      /Users/ziegler/Desktop/Release-1-tracking-results/multi_forward_0003.0.evio
1.4G      /Users/ziegler/Desktop/Release-1-tracking-results/multi_forward_0004.0.evio
1.4G      /Users/ziegler/Desktop/Release-1-tracking-results/multi_forward_0005.0.evio
1.4G      /Users/ziegler/Desktop/Release-1-tracking-results/multi_forward_0008.0.evio
Result:
2.4G      output.hipo
```

Benchmark

- **Benchmark summary**
  - uncompressed data is the same size as evil file.
  - writing speeds of LZ4 are far superior to GZIP
  - decompression speed (throughput) is much faster (~x7) for LZ4
  - LZ4 compression in fast more produces slightly larger files than GZIP







New Features

- CLAS12-RECONSTRUCTION
  - added HIPO support for reconstruction (experimental)
  - added output FILTERING for reconstruction (limited not configurable)
    - saves generated bank
    - saves reconstructed particles and detector response bank (compact)
  - ability to save DSTs for large data sets and fast analysis.

Benefits

- Compression of GEMC data files reduces storage space by 60% (compression 2.54).
- There is no limitation of 2GB per file.
- Easy access through HTTP, no need to read entire file to get events.
- File sharing through CLARA, cached partial indexing allows readout of any event.
- Indexing of data is 0.0029 sec for 2GB file, memory impact 12Kb.
  - (EVIO 2GB, indexing 30 sec, memory impact 1GB)
- Trying to put into standard EVIO library.

- **Impact on users**

Change to DataSource

```
EvioSource  reader = new EvioSource();
reader.open(inputFile);
EventDecoder decoder = new EventDecoder();
int icounter = 0;
```

Data Object Abstraction Model

- Java provides annotation framework implemented data description model
- Each data member is declared with annotation describing it’s properties
- EVIO handles annotations and automatically fills the class members.

```
public class DetectorBank {

    private List<Integer>  adcL = new ArrayList<Integer>();
    public DetectorBank(){}

    @EvioDataType(parent=1200,tag=1202,num=3,type="int32")
    public List getADCL(){
        return this.adcL;
    }
}
```

```
DetectorBank FTOFBank = new DetectorBank();

Evildataevent event = reader.getNextEvent();

 EvioObjectReader.readObject(event, FTOFBank);
```

- Writing to the EVIO will be as easy (not implemented yet)
- Can help us move away from Dictionaries

Online/Offline

- **Online produces EVIO file with compact banks:**
  - contains crate,slot and channel for each FADC data.
  - No translation is done.
  - No pedestal subtraction is done.
- **Offline Software framework:**
  - reads from database FADC NSA, NSB values.
  - reads from database translation tables.
  - implements compact structure decoder.
  - implements FADC pulse subtraction.
  - implements translation from CRATE/SLOT/CHANEL to DETECTOR/SECTOR/LAYER/COMPONENT.

- **Question:**
  - does this have to be part of OFFLINE software or ONLINE ?
  - Who developed the tools for the data decoding and translation ?

Calibration Constants

- Reading tables from database
- Parsing environment variables to see if user requests local database
- establish connection and read table
- translate table columns to appropriate values, like Integer, Float, String

- Reading tables from database
- Usage :

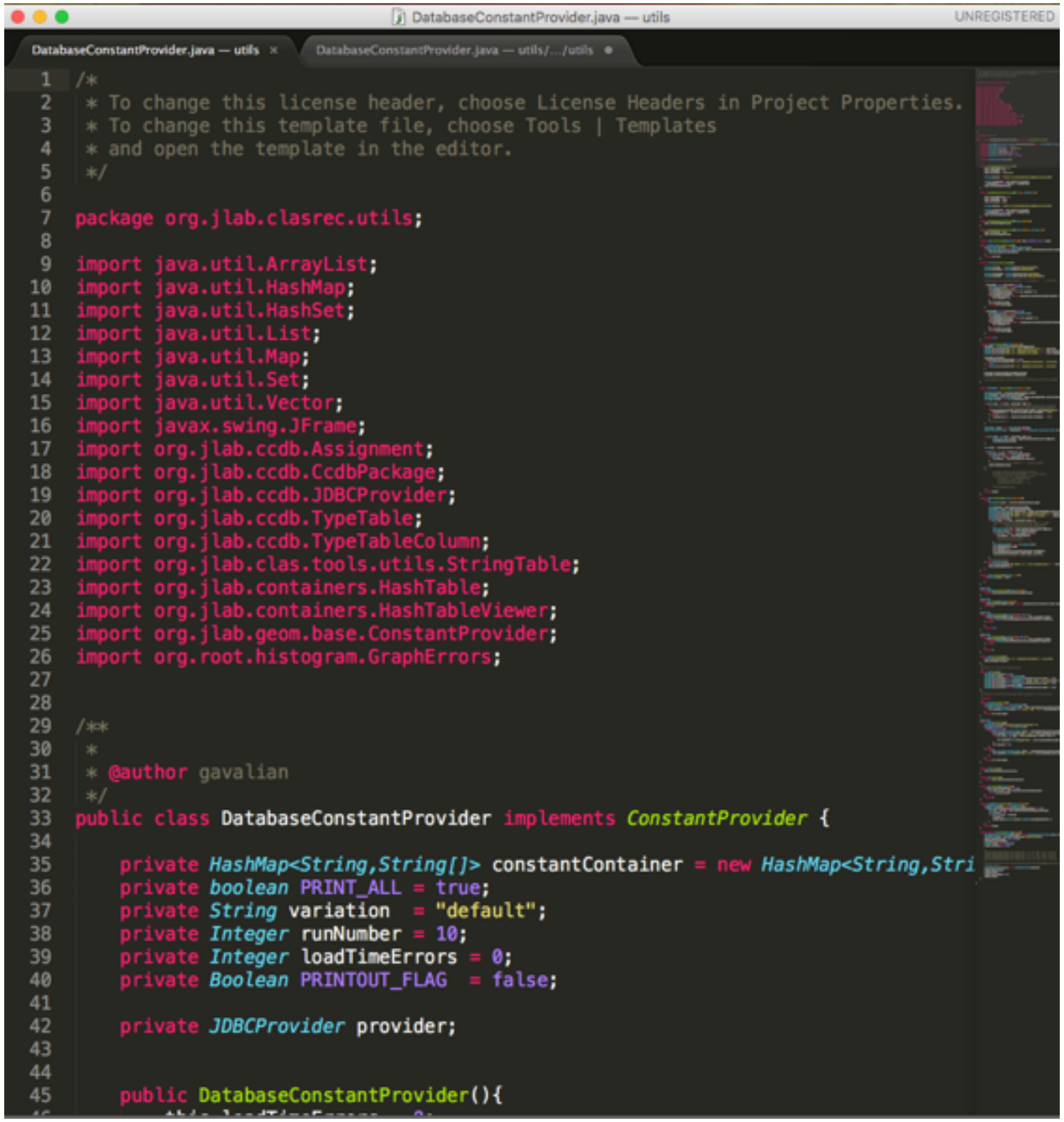
  DatabaseConstantProvider p(run,var);

  p.readTable("/calib/ec/atten");

- Analog to CLAS6:

```
map_get_float(def_map,map_layer,map_orient,EC_CHANNELS_PER_LAYER,values, *runno, &firsttime);
```

371 Lines



DatabaseConstantProvider.java — utils   UNREGISTERED

DatabaseConstantProvider.java — utils ×   DatabaseConstantProvider.java — utils/.../utils

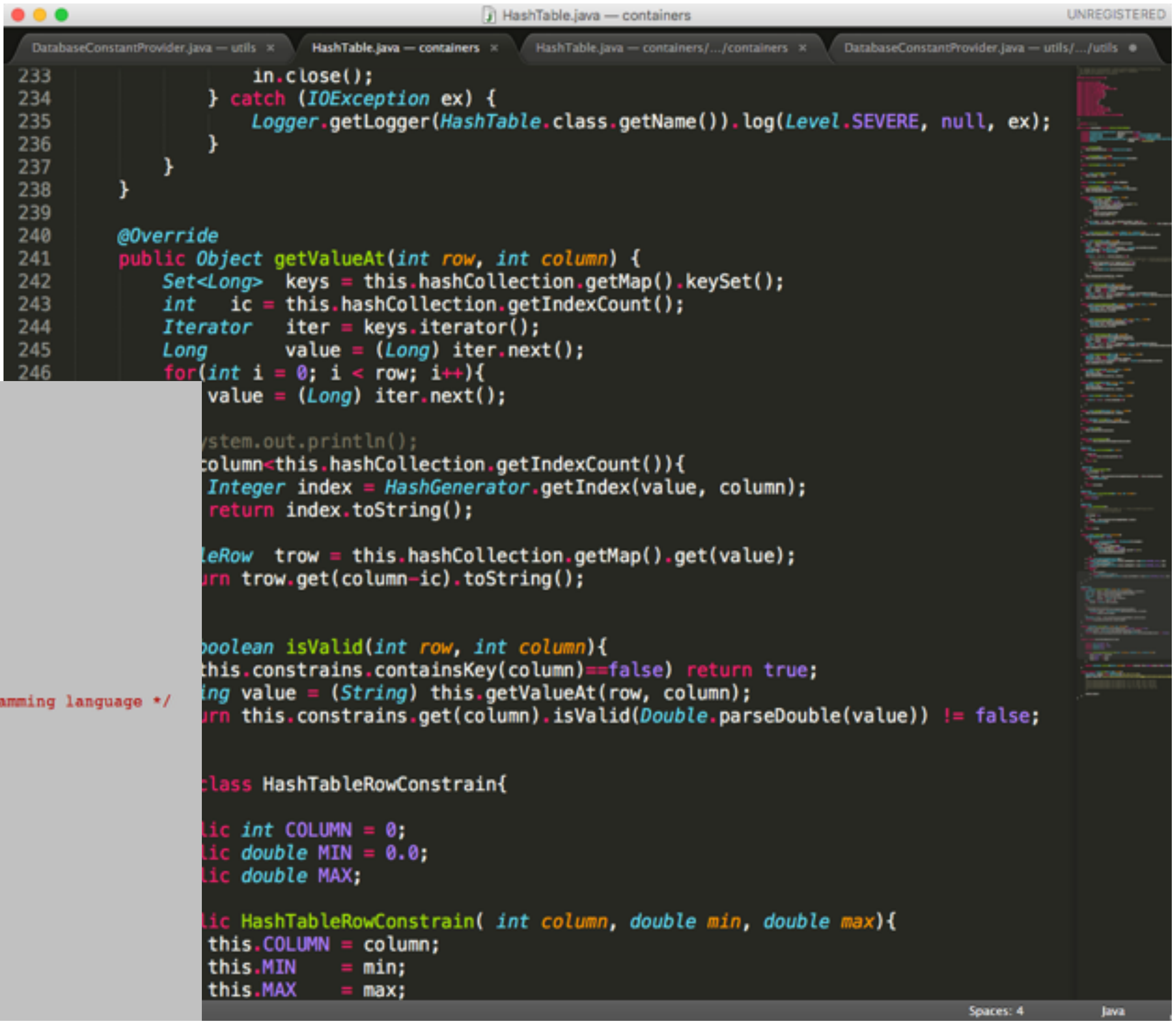
```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package org.jlab.clasrec.utils;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Vector;
import javax.swing.JFrame;
import org.jlab.ccdb.Assignment;
import org.jlab.ccdb.CcdbPackage;
import org.jlab.ccdb.JDBCProvider;
import org.jlab.ccdb.TypeTable;
import org.jlab.ccdb.TypeTableColumn;
import org.jlab.clas.tools.utils.StringTable;
import org.jlab.containers.HashTable;
import org.jlab.containers.HashTableViewer;
import org.jlab.geom.base.ConstantProvider;
import org.root.histogram.GraphErrors;


/**
 *
 * @author gavalian
 */
public class DatabaseConstantProvider implements ConstantProvider {

    private HashMap<String,String[]> constantContainer = new HashMap<String,Stri
    private boolean PRINT_ALL = true;
    private String variation  = "default";
    private Integer runNumber = 10;
    private Integer loadTimeErrors = 0;
    private Boolean PRINTOUT_FLAG  = false;

    private JDBCProvider provider;


    public DatabaseConstantProvider(){
```

Calibration Constants

- Hash Table for indexing constants by SECTOR, LAYER and COMPONENT

- CLAS-6 version for EC, all index arrays are hard coded
- **603 Lines of Code for EC**

291 Lines

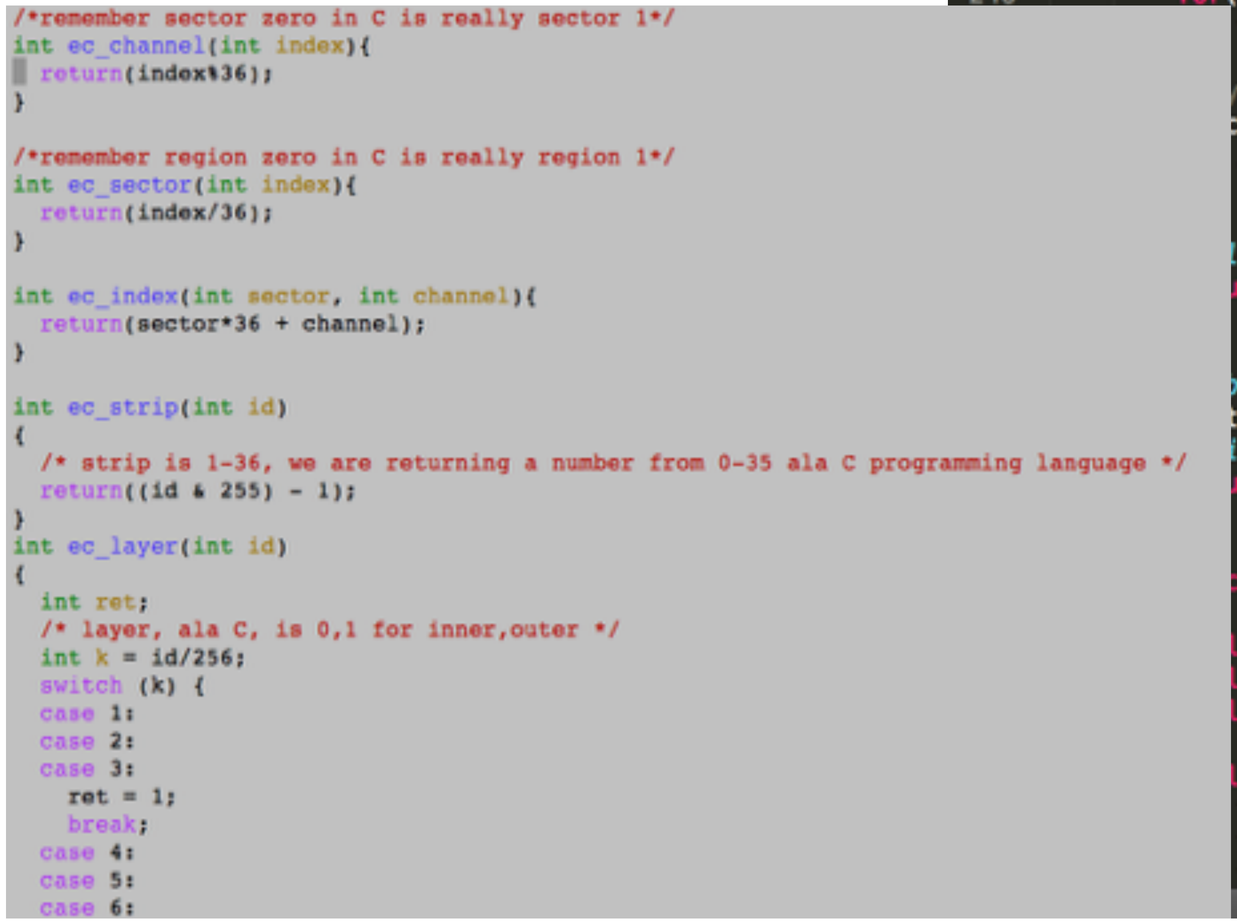


HashTable.java — containers   UNREGISTERED

DatabaseConstantProvider.java — utils | HashTable.java — containers | HashTable.java — containers/.../containers | DatabaseConstantProvider.java — utils/.../utils

```
                in.close();
            } catch (IOException ex) {
                Logger.getLogger(HashTable.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }

    @Override
    public Object getValueAt(int row, int column) {
        Set<Long>  keys = this.hashCollection.getMap().keySet();
        int    ic = this.hashCollection.getIndexCount();
        Iterator   iter = keys.iterator();
        Long       value = (Long) iter.next();
        for(int i = 0; i < row; i++){
            value = (Long) iter.next();

ystem.out.println();
column<this.hashCollection.getIndexCount()){
 Integer index = HashGenerator.getIndex(value, column);
 return index.toString();

leRow  trow = this.hashCollection.getMap().get(value);
urn trow.get(column-ic).toString();


boolean isValid(int row, int column){
this.constrains.containsKey(column)==false) return true;
ing value = (String) this.getValueAt(row, column);
urn this.constrains.get(column).isValid(Double.parseDouble(value)) != false;


class HashTableRowConstrain{

lic int COLUMN = 0;
lic double MIN = 0.0;
lic double MAX;

lic HashTableRowConstrain( int column, double min, double max){
 this.COLUMN = column;
 this.MIN    = min;
 this.MAX    = max;
```

Spaces: 4   Java



```
/*remember sector zero in C is really sector 1*/
int ec_channel(int index){
  return(index%36);
}

/*remember region zero in C is really region 1*/
int ec_sector(int index){
  return(index/36);
}

int ec_index(int sector, int channel){
  return(sector*36 + channel);
}

int ec_strip(int id)
{
  /* strip is 1-36, we are returning a number from 0-35 ala C programming language */
  return((id & 255) - 1);
}
int ec_layer(int id)
{
  int ret;
  /* layer, ala C, is 0,1 for inner,outer */
  int k = id/256;
  switch (k) {
  case 1:
  case 2:
  case 3:
    ret = 1;
    break;
  case 4:
  case 5:
  case 6:
```

Calibration Constants

- HashTable Helper Classes:
  - TableRow – responsible for keeping one indexed row
  - HashCollection – collection of indexed items
  - HashGenerator – generated hash code for mapping
  - HashTableViewer – displays the table allows search and value highlighting
  - IHashTableListener – callback for selected row in a hash table

- TOTAL 605 Lines of code (With everything 1267 Lines)
- This infrastructure used in all codes, and also for FADC parameter reading and Translation talbles.

- CLAS-6:
  - ec_maputil – 603 Lines
  - dc_maputil – 322 Lines
  - sc_maputil – 272 Lines
  - cc_maputil – 212 Lines
  - Total 1409 Lines of single purpose code
- code can not be used for any other detector or calibration

Story continues

- sc_calib/read_map.c – 210 Lines
- st_maputilities.c – 139 Lines
- dc_sigma.c – 103 Lines
- tagcal_read_map.F – 412 Lines
- tag_get_runcontrol.F – 152 Lines
- trk_maputil.c – 46 Lines
- stg_makeSTG – 118 Lines
- sc_geom.c – 459 Lines
- scaler/pflux_util.c – 714 Lines
- scaler/norm.c – 305 Lines
- taggercalib/get_tagger_map.c – 273 Lines
- gsim/dc/dc_maputil.c – 322 Lines
- ……….
- Total 3000+ Lines (only these),
- I found over 60 more routines that do custom indexing and reimplement code.

Detector Components

- **Detector Geometry:**
  - organized core parameters in the database
  - reads core parameters and constructs geometry objects that can be used by reconstruction code.
- **Calibration Constants:**
  - standardized detector constants table representation
  - automated indexing of constants by detector/sector/layer
  - automated comparison between runs and and versions

- **Question:**
  - Do we leave it up to a detector developer to read their constants and construct their detector ?
  - Do we leave it up to a detector groups to read the calibration constants and sort them an index them ?

Calibration Procedures

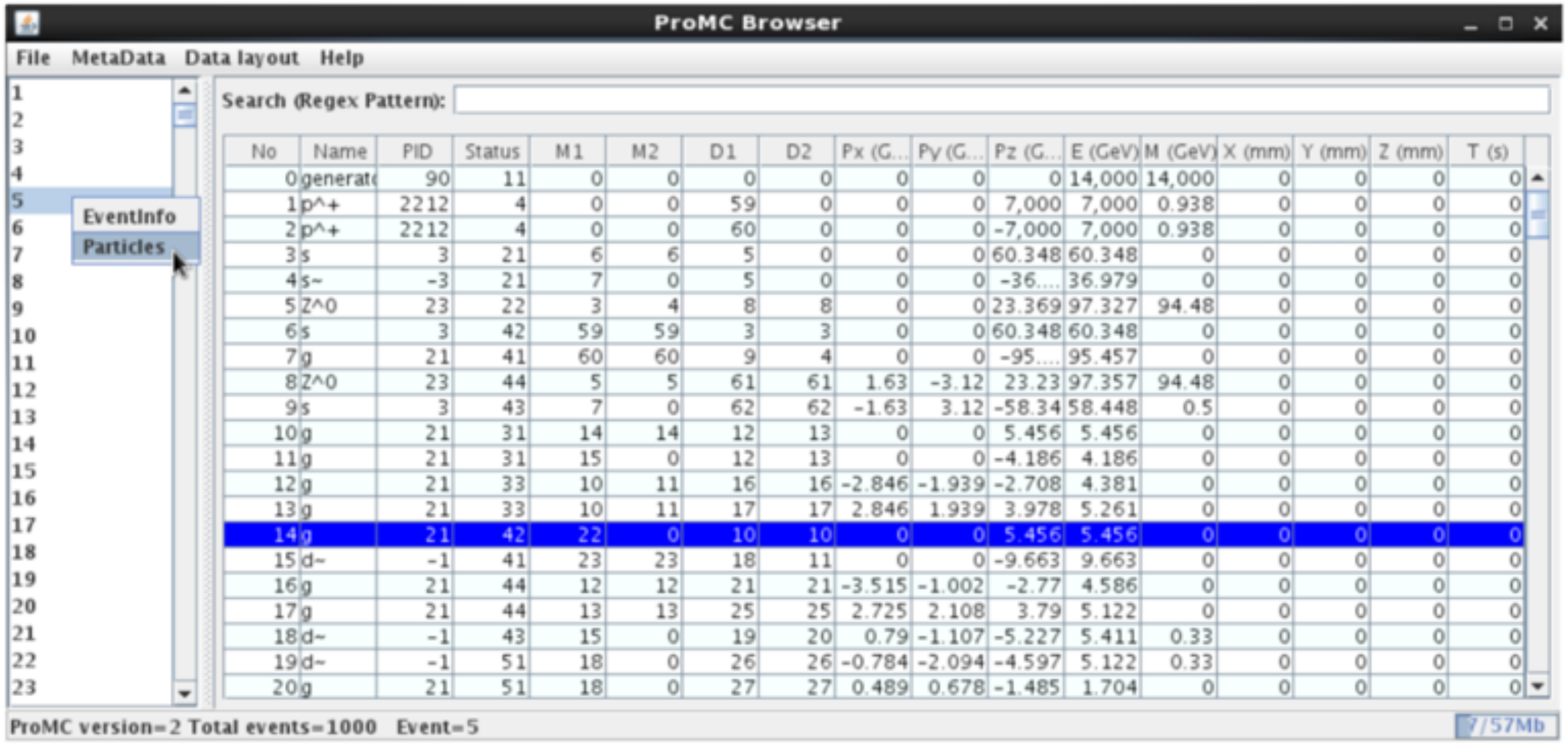
- **Calibration effort:**
  - developed a detector visualization framework by tying detector component clicking with Graph Canvas.
  - Implemented standard data processing interfaces and graphical tool to connect to a file or and ET ring. Made development of plugins REALLY EASY.

- **Question:**
  - Do we rely on developers to implement their own code to connect to ET ring and to read files ?
  - Who writes tools for ROOT/C++ to connect to ET and Files in similar fashion ?
  - Do we rely on developers to come up with their visualization of detectors and implement complicated GUI elements to be responsible for callbacks and data storage ?

Data Formats

- **Distributed Computing Era:**
  - I’m putting an effort into developing structures for EVIO for network readable files (through HTTP and CLARA).
  - big effort is going on to try tor develop object description model for data storage.
  - this request comes from detector groups (not my own initiative), I think it’s very important work.
- **Question:**
  - Do we need to abandon this project ?
  - How do we make files available through network ? What data format we use ? Who developed this ? Do we really need it ? (my opinion – “YES”)
  - is this ability present in other data formats ?
  - like ROOT ?
  - who investigates it ?
  - who writes interfaces ?

Limited knowledge



Search (Regex Pattern):

No	Name	PID	Status	M1	M2	D1	D2	Px (G...	Py (G...	Pz (G...	E (GeV)	M (GeV)	X (mm)	Y (mm)	Z (mm)	T (s)
0	generat	90	11	0	0	0	0	0	0	0	14,000	14,000	0	0	0	0
1	p^+	2212	4	0	0	59	0	0	0	7,000	7,000	0.938	0	0	0	0
2	p^+	2212	4	0	0	60	0	0	0	-7,000	7,000	0.938	0	0	0	0
3	s	3	21	6	6	5	0	0	0	60.348	60.348	0	0	0	0	0
4	s~	-3	21	7	0	5	0	0	0	-36....	36.979	0	0	0	0	0
5	Z^0	23	22	3	4	8	8	0	0	23.369	97.327	94.48	0	0	0	0
6	s	3	42	59	59	3	3	0	0	60.348	60.348	0	0	0	0	0
7	g	21	41	60	60	9	4	0	0	-95....	95.457	0	0	0	0	0
8	Z^0	23	44	5	5	61	61	1.63	-3.12	23.23	97.357	94.48	0	0	0	0
9	s	3	43	7	0	62	62	-1.63	3.12	-58.34	58.448	0.5	0	0	0	0
10	g	21	31	14	14	12	13	0	0	5.456	5.456	0	0	0	0	0
11	g	21	31	15	0	12	13	0	0	-4.186	4.186	0	0	0	0	0
12	g	21	33	10	11	16	16	-2.846	-1.939	-2.708	4.381	0	0	0	0	0
13	g	21	33	10	11	17	17	2.846	1.939	3.978	5.261	0	0	0	0	0
14	g	21	42	22	0	10	10	0	0	5.456	5.456	0	0	0	0	0
15	d~	-1	41	23	23	18	11	0	0	-9.663	9.663	0	0	0	0	0
16	g	21	44	12	12	21	21	-3.515	-1.002	-2.77	4.586	0	0	0	0	0
17	g	21	44	13	13	25	25	2.725	2.108	3.79	5.122	0	0	0	0	0
18	d~	-1	43	15	0	19	20	0.79	-1.107	-5.227	5.411	0.33	0	0	0	0
19	d~	-1	51	18	0	26	26	-0.784	-2.094	-4.597	5.122	0.33	0	0	0	0
20	g	21	51	18	0	27	27	0.489	0.678	-1.485	1.704	0	0	0	0	0

ProMC version=2 Total events=1000 Event=5 7/57Mb

Figure 2: A JAVA browser for ProMC files with events from a MC generator. The upper field is a search for a given particle name.

(12) Conversion to ROOT, HEPMC, HEPEVT, LHE, STDHEP, LCIO

One can convert ProMC file to ROOT to look at branches. If the ProMC package is installed, run the convertor:

```
cp -rf $PROMC/examples/promc2root .
cd promc2root
make
./promc2root [promc file] output.root
```

The output file will contain ROOT branches with px,py,pz,e, etc.

One can also convert ProMC to HEPMC using the example $PROMC/examples/promc2hepmc (see the ProMC manual). In additiion, the directory $PROMC/examples/ has examples showing how to convert ProMC to HEPEVT records (promc2hepevnt), STDHEP (promc2stdhep), LHE (promc2lhe) and LCIO (promc2lcio).

Data Visualization

- **Development of Data Visualization (jROOT):**
  - developed a visualization software for Java to graph histograms and graphs.
  - developed I/O for histograms and n-tuples from our Java framework.
  - developed GUI for browsing Histogram files and n-tuple with plotting capability.

- **Question:**
  - If this development is suspended, what do we use for data visualization ?
  - If there is no Java visualization and I/O, we have to convert our files to ROOT at the earliest stage, after the decoding and translating ?
  - Do we rely on users to find their own framework in Java to visualize data ?
  - Do we develop ROOT transition for each stage of our data processing ?

General Direction

- **Framework to handle data decoding and translation:**
  - presently in Java, can stay in java then data banks get transferred to ROOT.
  - we need tools in ROOT to read decoding tables.
- **Framework for Everything for Calibration:**
  - useless software like data visualization and Input/Output (mimicking ROOT).
  - useless detector visualization and mouse-over callback abilities.
  - useless framework for reading and writing Calibration constants.
  - geometry package, that describes all geometric shapes of detectors.
- **Framework for Reconstruction and Validation:**
  - reconstruction is Java

General Direction

- **What do we use for our Visualization Framework ?**
- **ROOT:**
  - need to implement root structures for storing RAW data.
  - need infrastructure to read from database calibration constants and FADC parameters.
  - abandon unnecessary overhead on developing Java utilities.
  - someone has to take this task and direct it's development.
  - UI elements have to me implemented for detector visualization.
  - detector geometry package has to be developed in C++.

- **To Proceed:**
  - Software group needs answers on all questions posted to continue development, and to not diverge from chosen path.

Release Status

- **Task List:**
  - write interface for loading fadc data and translation tables from database, create container that can cross reference pulse information (from fadc table), and load pulses and display, with information of detector and hardware.
  - write interface (API) for reading calibration constants for given detector for all variations and run numbers (or chose a range), and display the on top of each other, and ability to display ratio or difference for chosen reference constant set.
  - write library for constructing GEANT4 volumes from geometry database. currently our framework creates custom geometry shapes, we need GEANT4 volumes for all active volumes and passive volumes, define an interface for adding material properties to each volume element, and implement all material definitions. developed software has to also implement collision check capability and output generation for GEMC.
  - Continue development for online detector visualization and monitoring, extend framework to implement plugin structure that all detector groups can put their desired plots into the common visualization. this library needs to provide ability for plugins to report problems to the main framework, main framework should display warning and relate it to hardware information (translation table)
  - develop framework for storing bank definitions in the database, framework should include editing capability, versioning (CCDB should be used), reading, writing and validation all banks (check if same ID's or names have been used for more than one entry)