Hall-D File Formats

(groan)

David Lawrence JLab

Aug. 10, 2010

HDDM

- Started in 2001 by Richard Jones at UConn specifically for GlueX
- Data model driven by relationship of data, as opposed to code
- Data structures defined in XML then I/O routines made by auto-generating C code
- XML schema used for validation and run time version checks
- XDR routines (part of RPC) used for low-level I/O to provide platform independence
- Several utilities provided including XML<->HDDM converters

Top few lines of event.xml file used to define the current physics event data model for Hall-D

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<HDDM xmlns="http://www.gluex.org/hddm" class="s" version="1.0">

  <physicsEvent eventNo="int" maxOccurs="unbounded" runNo="int">
    <reaction maxOccurs="unbounded" minOccurs="0" type="int" weight="float">
      <beam minOccurs="0" type="Particle_t">
        <momentum E="float" px="float" py="float" pz="float"/>
        <properties charge="int" mass="float"/>
      </beam>
      <target minOccurs="0" type="Particle_t">
        <momentum E="float" px="float" py="float" pz="float"/>
        <properties charge="int" mass="float"/>
      </target>
      <vertex maxOccurs="unbounded">
        <product decayVertex="int" id="int" maxOccurs="unbounded" mech="int" p
          <momentum E="float" px="float" py="float" pz="float"/>
          <properties charge="int" mass="float"/>
        </product>
        <origin t="float" vx="float" vy="float" vz="float"/>
      </vertex>
    </reaction>
    <hitView minOccurs="0" version="2.0">
      <centralDC minOccurs="0">
        <cdcStraw maxOccurs="unbounded" minOccurs="0" ring="int" straw="int">
          <cdcStrawHit dE="float" maxOccurs="unbounded" t="float"/>
          <cdcStrawTruthHit dE="float" maxOccurs="unbounded" t="float"/>
        </cdcStraw>
```

XML vs. C++

event.xml

```
<hitView minOccurs="0" version="2.0">
  <centralDC minOccurs="0">
    <cdcStraw maxOccurs="unbounded" minOccurs="0" ring="int" straw="int">
      <cdcStrawHit dE="float" maxOccurs="unbounded" t="float"/>
      <cdcStrawTruthHit dE="float" maxOccurs="unbounded" t="float"/>
    </cdcStraw>
    <cdcTruthPoint dEdx="float" dradius="float" maxOccurs="unbounded" minOccurs="0" p
  </centralDC>
  <forwardDC minOccurs="0">
    <fdcChamber layer="int" maxOccurs="unbounded" module="int">
      <fdcAnodeWire maxOccurs="unbounded" minOccurs="0" wire="int">
        <fdcAnodeHit dE="float" maxOccurs="unbounded" t="float"/>
        <fdcAnodeTruthHit dE="float" maxOccurs="unbounded" t="float"/>
      </fdcAnodeWire>
      <fdcCathodeStrip maxOccurs="unbounded" minOccurs="0" plane="int" strip="int">
        <fdcCathodeHit maxOccurs="unbounded" q="float" t="float"/>
        <fdcCathodeTruthHit maxOccurs="unbounded" q="float" t="float"/>
      </fdcCathodeStrip>
      <fdcTruthPoint E="float" dEdx="float" dradius="float" maxOccurs="unbounded" mi
    </fdcChamber>
  </forwardDC>
  <startCntr minOccurs="0">
    <stcPaddle maxOccurs="unbounded" minOccurs="0" sector="int">
      <stcHit dE="float" maxOccurs="unbounded" t="float"/>
    </stcPaddle>
    <stcTruthPoint E="float" dEdx="float" maxOccurs="unbounded" minOccurs="0" phi="f
  </startCntr>
  <barrelEMcal minOccurs="0">
    <bcalCell layer="int" maxOccurs="4" minOccurs="0" module="int" sector="int">
```

DEventSourceHDDM.cc

```
//------------------
// Extract_DFDCHit
//------------------
jerror_t DEventSourceHDDM::Extract_DFDCHit(s_HDDM_t *hddm_s,  JFactory<DFDCHit> *factory)
{
    /// Copies the data from the given hddm_s structure. This is called
    /// from JEventSourceHDDM::GetObjects. If factory is NULL, this
    /// returns OBJECT_NOT_AVAILABLE immediately.

    if(factory==NULL)return OBJECT_NOT_AVAILABLE;

    vector<DFDCHit*> data;

    // Acquire the pointer to the physics events
    s_PhysicsEvents_t* allEvents = hddm_s->physicsEvents;
    if(!allEvents) {
      //throw JException("Attempt to get physics events from HDDM source failed.");
        return NOERROR;
    }

    for (unsigned int m=0; m < allEvents->mult; m++) {

        // Acquire the pointer to the overall hits section of the data
        s_HitView_t *hits = allEvents->in[m].hitView;

        if (hits == HDDM_NULL) {
          //throw JException("HDDM source has no hits.");
            continue;
        }

        if (hits->forwardDC == HDDM_NULL) {
          //throw JException("HDDM source has no forwardDC information.");
            continue;
        }

        if (hits->forwardDC->fdcChambers == HDDM_NULL) {
          // throw JException("HDDM source has no hits in the FDC.");
            continue;
        }

        // Acquire the pointer to the beginning of the FDC hit tree
        s_FdcChambers_t* fdcChamberSet = hits->forwardDC->fdcChambers;

        for (unsigned int i=0; i < fdcChamberSet->mult; i++) {
            // Each chamber in the ChamberSet has a wire set and a strip set
            s_FdcChamber_t &fdcChamber      = fdcChamberSet->in[i];
            s_FdcAnodeWires_t* wireSet      = fdcChamber.fdcAnodeWires;
            s_FdcCathodeStrips_t* stripSet  = fdcChamber.fdcCathodeStrips;

            // Each set of wires has (obviously) wires inside of it, and each wire
            // may have one or more hits on it. Make a DFDCHit object for each one
            // of these hits.
            for (unsigned int j=0; j < wireSet->mult; j++) {
                s_FdcAnodeWire_t anodeWire      = wireSet->in[j];
                s_FdcAnodeHits_t* wireHitSet    = anodeWire.fdcAnodeHits;
                for (unsigned int k=0; k < wireHitSet->mult; k++) {
                    s_FdcAnodeHit_t wireHit     = wireHitSet->in[k];
                    DFDCHit* newHit             = new DFDCHit();
                    newHit->layer               = fdcChamber.layer;
                    newHit->module              = fdcChamber.module;
                    newHit->element             = anodeWire.wire;
                    newHit->q                   = wireHit.dE;
                    newHit->t                   = wireHit.t;
                    newHit->plane               = 2;
                    newHit->type                = 0;
                    newHit->gPlane              = DFDCGeometry::gPlane(newHit);
                    newHit->gLayer              = DFDCGeometry::gLayer(newHit);
                    newHit->r                   = DFDCGeometry::getWireR(newHit);

                    data.push_back(newHit);
                }
            }

            // Ditto for the cathodes.
```

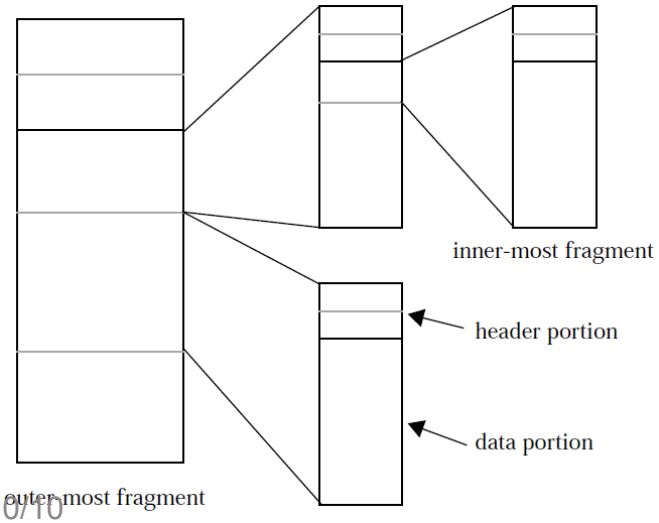
The auto-generated C data structures were linked into an tree using pointers to represent one event.

A data model elegantly expressed in XML became cumbersome to deal with as C-structures due to pointer checking and loops

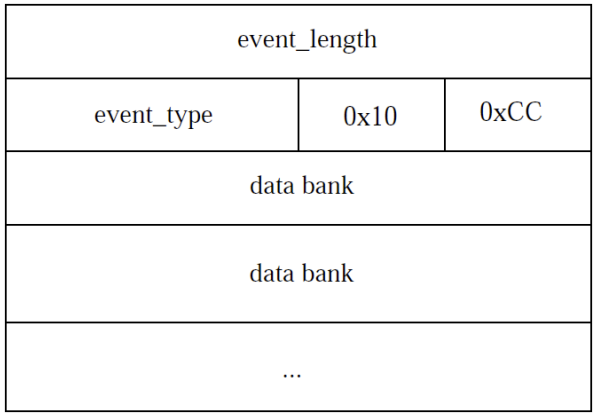
EVIO

- Developed by JLab DAQ group in early 1990s(?) (Elliott is maintainer)
- Designed for raw data: compact, platform independent, robust
- Events have minimal header (2, 4 or 8 bytes depending on application)
- Data stored in “banks” (arrays) with specific type
  - Atomic types (e.g. int, float, unsigned long ,…)
  - Bank of banks
  - All elements in a single bank are of same type
- Banks distinguished with pair of numbers (tag and num)
- Optional dictionary can be used with evio2xml for pretty printing

Event with fragment depth of three



Outermost bank of a physics event



Banks vs. C++

DTrackTimeBased.h

```
class DTrackTimeBased:public DKinematicData{
    public:
        JOBJECT_PUBLIC(DTrackTimeBased);

        oid_t trackid;          ///< id of DTrackW
        oid_t candidateid;   ///< id of DTrackCand
        float chisq;            ///< Chi-squared t
        int Ndof;               ///< Number of deg
        vector<DTrackFitter::pull_t> pulls; ///< H

        const DReferenceTrajectory *rt; ///< point

        double FOM;
```

DEventSourceEVIO.cc

```
    // Get list of DTrackTimeBased banks. At this level, there should usually be
    // only one but we get the list and loop over them since that is how it is formatted.
    // The actual DTrackTimeBased objects are contained in child banks of this (these).
    evioDOMNodeListP bList = evt->getNodeList(tagNumEquals(tagMap["DTrackTimeBased"]));

    // loop over all banks in list (should only be one)
    evioDOMNodeList::const_iterator iter1;
    for(iter1=bList->begin(); iter1!=bList->end(); iter1++) {

        evioDOMNodeList *members = (*iter1)->getChildList();

        try{
            const vector<uint64_t> &v_objId = GetVector<uint64_t>(members, "DTrackTimeBased.objId");
            const vector<float> &v_chisq = GetVector<float>(members, "DTrackTimeBased.chisq");
            const vector<int> &v_Ndof = GetVector<int>(members, "DTrackTimeBased.Ndof");
            const vector<float> &v_FOM = GetVector<float>(members, "DTrackTimeBased.FOM");
            const vector<float> &v_x = GetVector<float>(members, "DTrackTimeBased.x");
            const vector<float> &v_y = GetVector<float>(members, "DTrackTimeBased.y");
            const vector<float> &v_z = GetVector<float>(members, "DTrackTimeBased.z");
            const vector<float> &v_px = GetVector<float>(members, "DTrackTimeBased.px");
            const vector<float> &v_py = GetVector<float>(members, "DTrackTimeBased.py");
            const vector<float> &v_pz = GetVector<float>(members, "DTrackTimeBased.pz");
            const vector<float> &v_q = GetVector<float>(members, "DTrackTimeBased.q");
            //const vector<float> &v_E = GetVector<float>(members, "DTrackTimeBased.E");
            const vector<float> &v_mass = GetVector<float>(members, "DTrackTimeBased.mass");
            //const vector<float> &v_t0 = GetVector<float>(members, "DTrackTimeBased.t0");

            // Get enough DReferenceTrajectory objects for all of the DTrackTimeBased Objects
            // we're about to read in. This seems a little complicated, but that's because it
            // is expensive to allocate these things so we recycle as much as possible.
            list<DReferenceTrajectory*> my_rts;
            pthread_mutex_lock(&rt_mutex);
            while(my_rts.size() < v_objId.size()){
                if(rt_pool.size()>0){
                    my_rts.push_back(rt_pool.back());
                    rt_pool.pop_back();
                }else{
                    my_rts.push_back(new DReferenceTrajectory(bfield));
                }
            }
            pthread_mutex_unlock(&rt_mutex);

            // Loop over DTrackTimeBased objects
            event_had_tracktimebaseds = true;
            for(unsigned int i=0; i<v_objId.size(); i++){

                DVector3 pos(v_x[i], v_y[i], v_z[i]);
                DVector3 mom(v_px[i], v_py[i], v_pz[i]);

                DTrackTimeBased *track = new DTrackTimeBased();

                track->setMomentum(mom);
                track->setPosition(pos);
                track->setCharge(v_q[i]);
                track->setMass(v_mass[i]);
                track->chisq = v_chisq[i];
                track->Ndof = v_Ndof[i];
                track->FOM = v_FOM[i];
                track->id = v_objId[i];
```

Single-type banks requires a "barrel" transformation to store structures of mixed type in EVIO.

Reading in requires user to allocate data structure and copy elements into it.

Why make a HDDM to EVIO converter?

- DAQ system will record data in EVIO format
- Monitoring and analysis software used for commissioning will need to be exercised prior to DAQ system data being available
- Simulated data MUST be converted to EVIO format for this purpose
- The *ded* event display needs this now because:
  - It is based on bCNU which already has JAVA-based EVIO capability
  - We have free student labor currently working on it

Why not make *hdgeant* write out EVIO formatted data?

- Simple Cost-benefit analysis:
  - Benefit:
    - HDDM is already well-integrated into *hdgeant* and offline already capable of reading HDDM
    - No direct benefit from switching
    - Indirect benefit by *potentially* avoiding future cost of maintaining multiple formats
  - Cost:
    - Switching costs manpower to implement
    - Maintainer of *hdgeant* will no longer be maintainer of file format

Bottom line: Definite cost + Potential benefit = Don't do it!*

**From risk management viewpoint, potential cost is a low-risk event due to its low cost and the ability to reverse the decision at any time*

Why File Format is mostly a non-issue

- Most of the software written for Hall-D will be based on C++ objects in memory.
- Only under rare and special circumstances would anyone need to write a piece of code with an “openFile” or “readEvent” type call.
- Framework was designed from day 1 to accommodate multiple formats so that ALL DANA programs would be file format agnostic

Arguments for switching to a single format

- Unsound:
  - If we were starting “green field” we would design it that way
    - (decisions based on non-existent conditions do not apply)
  - I like format “A” better than format “B”
    - (preferences do not equate to resources)
  - Things would seem simpler with just one format so I would feel better about it
- Invalid:
  - Lots of time will be saved by not running converter programs
    - (most use cases do not require converters)
  - Manpower will be saved by not maintaining multiple formats
    - (currently not costing much and decision can always be reversed)

Hall-D Data Flow