Hall A Analyzer 1.6 Overview

Ole Hansen

Jefferson Lab

Tritium Analysis Meeting
April 4, 2017

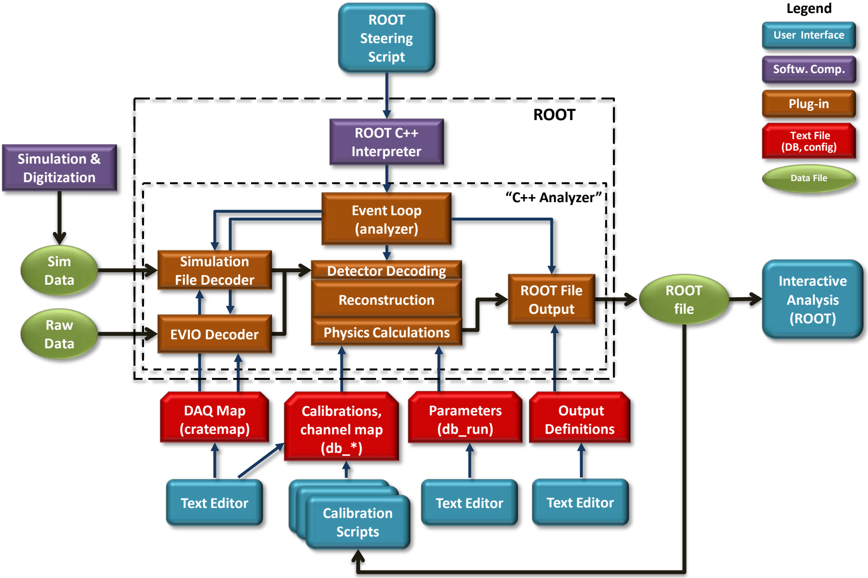
The Hall A C++ Analyzer (“Podd”)

- Class library on top of ROOT
- Provides modules for standard analysis tasks and Hall A equipment, *e.g.* HRS with VDCs
- Special emphasis on modularity
  - “Everything is a plug-in”
  - External user libraries dynamically loadable at run time
  - User code separate from core analyzer
  - Users should have to write only experiment-specific code
  - SDK for rapid development of new module libraries
- Developed jointly with Hall C since 2012
- Plenty of documentation at `http://hallaweb.jlab.org/podd/doc/`

Podd: Stengths, Limitations

- Strengths
  - Light-weight: minimal dependencies, small memory footprint
  - Apparently quite user-friendly: students learn easily
  - Output & cuts configurable at run-time via text files. Flat text file database
  - Full ROOT 6 support. Supported on Linux and Mac
- Limitations
  - Designed for one-pass analysis only: EVIO raw data $\rightarrow$ ROOT ntuple-style trees + histograms
  - Single-Threaded (multi-threaded version in development). Multi-threading not compelling for HRS analysis (I/O-bound)

Hall A C++ Analyzer Framework



Status

- Stable version: 1.5.37 (03-Mar-2017) web
  - Bugfixes
  - 1.5.x releases are binary-compatible
- Development version: 1.6-beta3 (19-Jan-2017) web
  - New database format
  - Many new features (see next), not all fully implemented/tested yet.
  - Hope to finalize by summer 2017 for fall run
  - Preliminary Release Notes available web
- Repository GitHub
  - For experts. Things may change unexpectedly.
  - Download:

    ```
    git clone https://github.com/JeffersonLab/analyzer.git
    ```

Version 1.6: Completed Items

- Modular decoder (Bob Michaels)
- Simulation event data decoder API
- EVIO (CODA file I/O) loaded from external library; to use, either
  - pre-install on your system; or
  - download & build automatically
- Miscellaneous
  - `scons` build system
  - Improved formula & test package (removed limitations)
  - Rewritten, modular hardware channel decoder (`THaDecData`)
  - Many small code improvements (see GitHub/ChangeLog)

Version 1.6: Work In Progress

- Generalized database interface
  - All analysis modules now use this interface.
  - Users must convert their existing databases. Conversion utility program available (`utils/dbconvert`, largely complete) ▸ doc
- Improved VDC track reconstruction
  - Known bugs fixed
  - Reconstruction of multi-cluster events should be greatly improved.
  - Needs testing/optimization. Testers welcome.
  - Old code will remain available as an alternative tracking algorithm.
- More efficient output module (data types, array size variables, etc.)
- Test suite
  - "`make test`" to check for code regressions
  - Standard practice, employed by most modern software packages, but time-consuming to implement

Database Format Conversion

Old Fixed-Format Database db_L.s1.dat

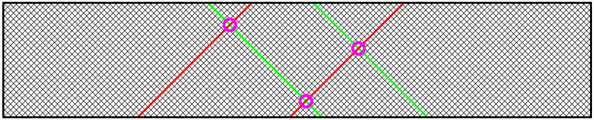
```
Number of Left Scintillator 1 paddles ---------------------
       6
Crate,Slot,1st,Last ADC chans,Beg S1 chan, model ----------------
       3      18      6       11      1       1881 - ADCs pads 1-6  (right)
       3      18      0       05      7       1881 - ADCs pads 7-12 (left)
       3      10      88      93      1       1877 - TDCs pads 1-6  (right)
       3      10      80      85      7       1877 - TDCs pads 7-12 (left)
      -1       0       0       0      0
X,Y,Z coords (in m) of S1 front plane in spectrom cs --------------
   -0.129     0.0    1.2873                             - Meters
Half of X, half of Y, full Z sizes (in m) of S1 ----------------
    0.88    0.18     0.005                              - Meters
TDC time offsets of S1 in TDC channels ---------------------
     2.45 6.38 7.58 3.78 -13.25 3.75                    - Left Paddles
    -14.13 -16.83 -0.40 -3.78 -22.70 -0.12              - Right Paddles
```

`dbconvert DB-old/ DB-new/` (converts entire directory) $\longrightarrow$

New Free-Format Key-Value Database

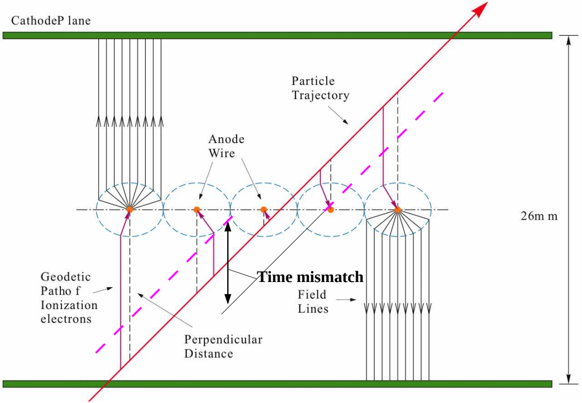
```
----[ 2015-03-24 00:00:00 -0400 ]
L.s1.detmap =  3    18     6    11     1  1881
               3    18     0     5     7  1881
               3    10    88    93     1  1877
               3    10    80    85     7  1877
L.s1.npaddles = 6
L.s1.position = -0.1290  0.0000  1.2873
L.s1.size = 0.88  0.18  0.005
L.s1.L.off = 2.45  6.38  7.58  3.78  -13.25  3.75
L.s1.R.off = -14.13  -16.83  -0.40  -3.78  -22.70  -0.12
```

VDC Reconstruction Challenge: Multiple Clusters



- With only two readout coordinates, ambiguities from multiple clusters cannot be reliably resolved.
- This is an inherent design limitation of the VDCs
- Try to make the best of it in software
  - Fit tracks through all possible combinations of clusters
  - Sort by χ^2 or similar goodness-of-fit criterion
  - Pick track with best χ^2, mark clusters from this combination used
  - Keep best track(s)
  - Use other detectors to remove obvious ghost tracks
  - Advanced: 3-parameter fit to determine common drift time offset
  - Advanced: analyze clusters for overlaps, noise hits
- Ideal: Add additional hardware to detector stack (*e.g.* FPP, 3rd VDC coordinate) to help resolve ambiguities

3-Parameter Cluster Fit



- Non-linear 3-parameter fit to extract track time offset t_0
- Computationally expensive: ca. $\times$20 slower than 2-parameter fit
- $\approx$ 20 ns FWHM time resolution $\rightarrow$ background rejection factor $\approx$ 10-20
- Important for high-rate experiments (APEX) with actual accidental coincidences

VDC Algorithm Improvements

Version 1.5.38

- Disallow UV ambiguities (configurable)
- UV fiducial cut
- Proper lower-upper matching cut
- Disallow cluster sharing

→ Guarantees clean single track at expense of slightly lower tracking efficiency

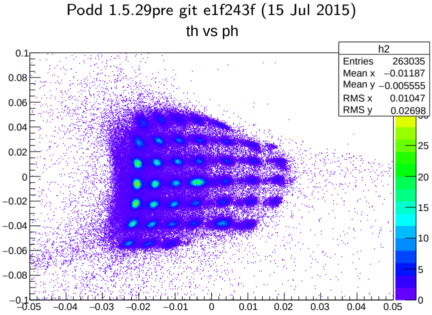
Version 1.6

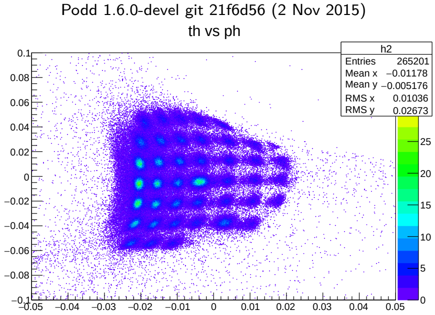
- Cluster shape analysis
- Overlapping cluster splitting (to do)
- 3-parameter cluster fit (to do)
- Cluster t_0 cut
- UV fiducial cut
- Proper lower-upper matching cut
- Disallow cluster sharing
- Old VDC code for reference (to do)

→ Allows multi-tracks, improves tracking efficiency, high-rate capable

VDC Reconstruction Comparison

313476 physics events from `/work/work/halla/g2p/disk1/ole/g2p_3132.dat.0`
(12 Mar 2012, LHRS @ 2.228 GeV, thin carbon foil, sieve slit, septum),
280231 successfully reconstructed tracks.





Testing & characterization needed

Learning More: Summer 2017 Analysis Workshop

- Dates June 26–27, 2017
- Topics under consideration
  - `Podd/hcana` introduction & news
  - Hands-on tutorials w/ example replays
  - Collected wisdom for setting up the software for a new experiment
  - ROOT Tips & Tricks
  - Calibration/optics how-tos
  - Introduction to JLab scicomp resources & batch farm
- Announcement later this week

Resources

- Web site ▸ home page
  - Documentation
  - Release Notes
  - Software Development Kit (SDK)
  - Source code downloads
  - Archived tutorials & example replays
- Bug tracker ▸ GitHub
- Bi-weekly Hall A/C software meeting: Wednesdays, 10am, F224/225
- Mailing list: `halla_software@jlab.org`. Subscribe on ▸ mailman
- Analysis Workshop archive ▸ archive (includes tutorials)

Closer Look

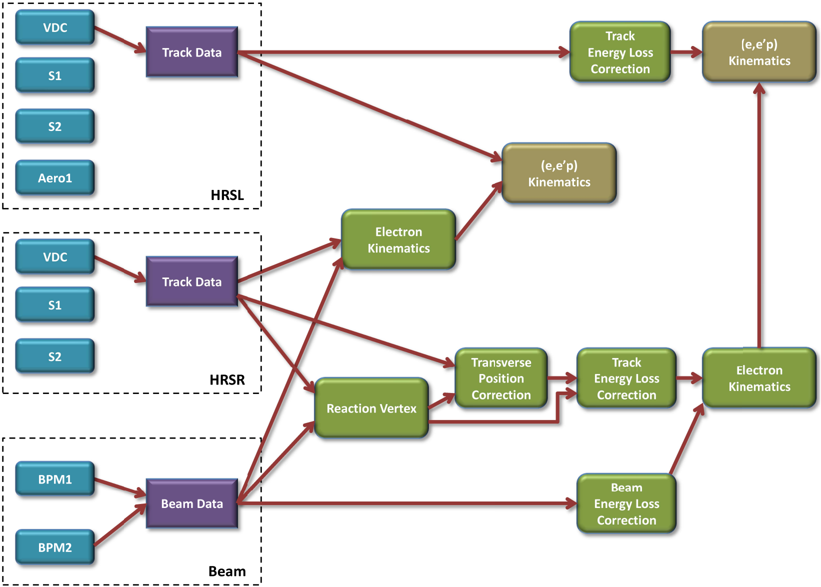
Analysis Objects

- Any module that produces "results"
- Every analysis object has unique name, *e.g.* R.s1
- Results stored in **"global variables"**, prefixed with the respective module's name, *e.g.* R.s1.nhits
- `THaAnalysisObject` common base class:
  - Support functions for database access
  - Support functions for global variable handling
- Actual objects implement various virtual functions
  - `DefineVariables()`
  - `ReadDatabase()`
  - `Decode()`
  - etc.

Types of Analysis Objects

- "Detector"
  - Code/data for analyzing a type of detector.
    Examples: Scintillator, Cherenkov, VDC, BPM
  - Typically embedded in an Apparatus
- "Apparatus" / "Spectrometer"
  - Collection of Detectors
  - Combines data from detectors
  - "Spectrometer": Apparatus with support for tracks
- "Physics Module"
  - Combines data from several apparatuses
  - Typical applications: kinematics calculations, vertex finding, coincidence time extraction
  - Toolbox design: Modules can be chained, combined, used as needed

A (complex) Module Configuration Example



Things You'll Need

1. Replay script
   - Defines detectors/apparatuses to be analyzed, kinematics, calculations to be done, file locations, tree variable names etc.
   - Many examples available from previous experiments
   - Simple or fancy ▸ fancy example. Try to start out simple
   - May be compiled
2. Set of database files
   - Usually one file per detector, `db_<name>.dat`
   - Run database, `db_run.dat`, defines beam energy, spectrometer angles
   - `db_cratemap.dat` and `scaler.map`, define decoder parameters
     → get these files from DAQ expert
3. Output definition file
   - Defines which variables to write to the tree in the output ROOT file
4. Raw data (CODA files)

C++ Analyzer User Interface

Example Replay Script (simplified, for ROOT 5)

```
// Set up right arm HRS with the detectors we're interested in
THaApparatus* HRSR = new THaHRS("R", "Right HRS");
HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") );
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Preshower"));
HRSR->AddDetector( new THaShower("sh", "Shower"));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaApparatus* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
gHaPhysics->Add( new THaGoldenTrack("gold","HRSR Golden Track","R") );
gHaPhysics->Add( new THaElectronKine("EKR","Electron kinematics R","R",mass_tg) );
gHaPhysics->Add( new THaReactionPoint("rpr","Reaction vertex R","R","IB") );

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/run_12345.root");  // Set output destination
analyzer->SetOdefFile("HRSR.odef");             // Define output
analyzer->Process(run);                         // Process all events in the input
```

C++ Analyzer User Interface

Example Replay Script (simplified, for ROOT 5)

```
// Set up right arm HRS with the detectors we're interested in
THaApparatus* HRSR = new THaHRS("R", "Right HRS");
HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") );
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Preshower"));
HRSR->AddDetector( new THaShower("sh", "Shower"));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaApparatus* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
gHaPhysics->Add( new THaGoldenTrack("gold","HRSR Golden Track","R") );
gHaPhysics->Add( new THaElectronKine("EKR","Electron kinematics R","R",mass_tg) );
gHaPhysics->Add( new THaReactionPoint("rpr","Reaction vertex R","R","IB") );

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/run_12345.root");  // Set output destination
analyzer->SetOdefFile("HRSR.odef");            // Define output
analyzer->Process(run);                        // Process all events in the input
```

C++ Analyzer User Interface

Example Replay Script (simplified, for ROOT 5)

```
// Set up right arm HRS with the detectors we're interested in
THaApparatus* HRSR = new THaHRS("R", "Right HRS");
HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") );
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Preshower"));
HRSR->AddDetector( new THaShower("sh", "Shower"));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaApparatus* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
gHaPhysics->Add( new THaGoldenTrack("gold","HRSR Golden Track","R") );
gHaPhysics->Add( new THaElectronKine("EKR","Electron kinematics R","R",mass_tg) );
gHaPhysics->Add( new THaReactionPoint("rpr","Reaction vertex R","R","IB") );

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/run_12345.root");  // Set output destination
analyzer->SetOdefFile("HRSR.odef");            // Define output
analyzer->Process(run);                        // Process all events in the input
```

C++ Analyzer User Interface

Example Replay Script (simplified, for ROOT 5)

```
// Set up right arm HRS with the detectors we're interested in
THaApparatus* HRSR = new THaHRS("R", "Right HRS");
HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") );
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Preshower"));
HRSR->AddDetector( new THaShower("sh", "Shower"));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaApparatus* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
gHaPhysics->Add( new THaGoldenTrack("gold","HRSR Golden Track","R") );
gHaPhysics->Add( new THaElectronKine("EKR","Electron kinematics R","R",mass_tg) );
gHaPhysics->Add( new THaReactionPoint("rpr","Reaction vertex R","R","IB") );

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/run_12345.root");  // Set output destination
analyzer->SetOdefFile("HRSR.odef");             // Define output
analyzer->Process(run);                         // Process all events in the input
```

C++ Analyzer User Interface

Example Replay Script (simplified, for ROOT 5)

```
// Set up right arm HRS with the detectors we're interested in
THaApparatus* HRSR = new THaHRS("R", "Right HRS");
HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") );
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Preshower"));
HRSR->AddDetector( new THaShower("sh", "Shower"));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaApparatus* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
gHaPhysics->Add( new THaGoldenTrack("gold","HRSR Golden Track","R") );
gHaPhysics->Add( new THaElectronKine("EKR","Electron kinematics R","R",mass_tg) );
gHaPhysics->Add( new THaReactionPoint("rpr","Reaction vertex R","R","IB") );

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/run_12345.root");  // Set output destination
analyzer->SetOdefFile("HRSR.odef");            // Define output
analyzer->Process(run);                        // Process all events in the input
```

C++ Analyzer User Interface

Example Replay Script (simplified, for ROOT 5)

```
// Set up right arm HRS with the detectors we're interested in
THaApparatus* HRSR = new THaHRS("R", "Right HRS");
HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") );
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Preshower"));
HRSR->AddDetector( new THaShower("sh", "Shower"));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaApparatus* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
gHaPhysics->Add( new THaGoldenTrack("gold","HRSR Golden Track","R") );
gHaPhysics->Add( new THaElectronKine("EKR","Electron kinematics R","R",mass_tg) );
gHaPhysics->Add( new THaReactionPoint("rpr","Reaction vertex R","R","IB") );

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/run_12345.root");  // Set output destination
analyzer->SetOdefFile("HRSR.odef");            // Define output
analyzer->Process(run);                        // Process all events in the input
```

Output Definitions

- Choose “global variables” to include in ROOT output tree
- Tree branches can be dynamically defined for each replay via input file

Example Output Definition File

```
# All variables from the GoldenTrack module
block R.gold.*

# Calculated quantities for inclusive electron scattering measured
# by the RHRS, form the ElectronKine physics module
block R.EKR.*

# All RHRS track data (focal plane as well as reconstructed to target)
# NOTE: Probably writes a lot of info you don't care about
block R.tr.*
```

- Much more possible
  - Arithmetic expressions
  - Using/defining cuts
  - 1D and 2D histograms
  - EPICS variables
  - Scalers
- Full documentation on the web ▸ docs (Bob Michaels)

Global Variable Definitions

Example DefineVariables() Function

```
Int_t THaPrimaryKine::DefineVariables( EMode mode ) {
  // Define/delete global variables.
  if( mode == kDefine && fIsSetup ) return kOK;
  fIsSetup = ( mode == kDefine );

  RVarDef vars[] = {
    { "Q2",      "4-momentum transfer squared (GeV^2)",     "fQ2" },
    { "omega",   "Energy transfer (GeV)",                   "fOmega" },
    { "W2",      "Invariant mass of recoil system (GeV^2)", "fW2" },
    { "angle",   "Scattering angle (rad)",                  "fTheta" },
    { "epsilon", "Virtual photon polarization factor",      "fEpsilon" },
    { "q3m",     "Magnitude of 3-momentum transfer",        "fQ3mag" },
    { "th_q",    "Theta of 3-momentum vector (rad)",        "fThetaQ" },
    { "ph_q",    "Phi of 3-momentum vector (rad)",          "fPhiQ" },
    { "nu",      "Energy transfer (GeV)",                   "fOmega" },
    { "q_x",     "x-cmp of Photon vector in the lab",       "fQ.X()" },
    { "q_y",     "y-cmp of Photon vector in the lab",       "fQ.Y()" },
    { "q_z",     "z-cmp of Photon vector in the lab",       "fQ.Z()" },
    { 0 }
  };
  return DefineVarsFromList( vars, mode );
}
```

Database

- Currently only flat text files supported
- Key/value pairs with support for scalars, arrays, matrices, strings
- Support for time-dependent values (essential!)
- History functionality available if files kept under version control (*e.g.* `git`)

Example Database File

```
B.mwdc.planeconfig = u1 u1p x1 x1p v1 v1p
                     u2 x2 v2
                     u3 u3p x3 x3p v3 v3p

# "Crate map":  crate slot_lo slot_hi   model# resol   nchan
B.mwdc.cratemap =  3     6       21       1877   500     96
                   4     4       11       1877   500     96
                   4     17      24       1877   500     96

--[ 2008-03-31 23:59:45 ]
B.mwdc.maxslope      = 2.5

B.mwdc.size          = 2.0  0.5   0.0
B.mwdc.x1.size       = 1.4  0.35  0.0
```

Database

- Currently only flat text files supported
- Key/value pairs with support for scalars, arrays, matrices, strings
- Support for time-dependent values (essential!)
- History functionality available if files kept under version control (*e.g.* `git`)

Example Database File

```
B.mwdc.planeconfig = u1 u1p x1 x1p v1 v1p
                     u2 x2 v2
                     u3 u3p x3 x3p v3 v3p

# "Crate map":  crate slot_lo slot_hi   model# resol   nchan
B.mwdc.cratemap =  3     6       21      1877   500     96
                   4     4       11      1877   500     96
                   4    17       24      1877   500     96

--[ 2008-03-31 23:59:45 ]
B.mwdc.maxslope      = 2.5

B.mwdc.size          = 2.0  0.5   0.0
B.mwdc.x1.size       = 1.4  0.35  0.0
```

Tests & Cuts

- `THaCut` & `THaCutList` ▸ doc
- Optional termination of analysis of current event at various stages
- Inherits from `TFormula` → broad range of expressions supported

Example Cut Definition File

```
Block: RawDecode

evtyp1            g.evtyp==1          // Event type 1 (=HRSR main trigger)
poshel            g.helicity==1
neghel            g.helicity==-1
goodhel           poshel||neghel
RawDecode_master  evtyp1

Block: Decode

NoisyU1           R.vdc.u1.nhit>50
NoisyV1           R.vdc.v1.nhit>50
NoisyU2           R.vdc.u2.nhit>50
NoisyV2           R.vdc.v2.nhit>50
NoisyVDC          NoisyU1||NoisyV1||NoisyU2||NoisyV2
EnoughShowerHits  R.sh.nhit>10
Decode_master     !NoisyVDC
```

Tests & Cuts

- THaCut & THaCutList ▸ doc
- Optional termination of analysis of current event at various stages
- Inherits from TFormula → broad range of expressions supported

Example Cut Definition File

```
Block: RawDecode

evtyp1              g.evtyp==1          // Event type 1 (=HRSR main trigger)
poshel              g.helicity==1
neghel              g.helicity==-1
goodhel             poshel||neghel
RawDecode_master    evtyp1

Block: Decode

NoisyU1             R.vdc.u1.nhit>50
NoisyV1             R.vdc.v1.nhit>50
NoisyU2             R.vdc.u2.nhit>50
NoisyV2             R.vdc.v2.nhit>50
NoisyVDC            NoisyU1||NoisyV1||NoisyU2||NoisyV2
EnoughShowerHits    R.sh.nhit>10
Decode_master       !NoisyVDC
```