

# Introduction to the Hall A Analyzer Framework

Ole Hansen

Jefferson Lab

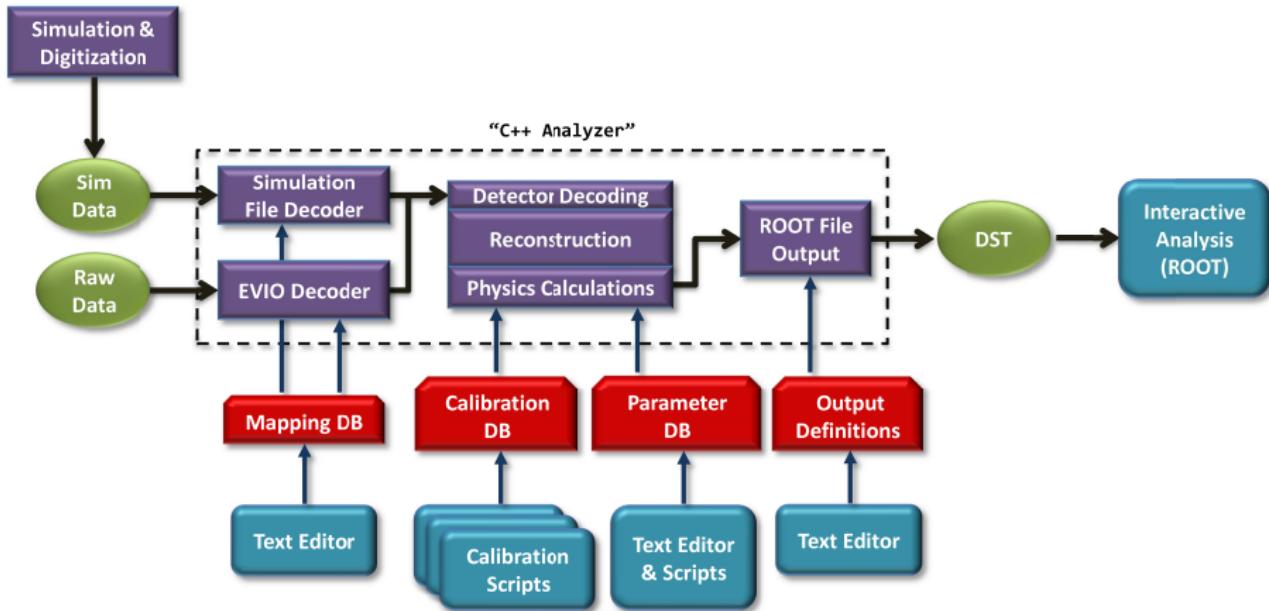
Hall C Summer Workshop  
June 22, 2012

<http://hallaweb.jlab.org/podd/>

# Contents

- 1 Software Framework Overview
- 2 Component Details & Examples
- 3 Status & Plans
- 4 For Developers
- 5 Summary

# Physics Replay Components



# C++ Analyzer (“Podd”) Overview

- Standard Hall A analysis software since 2003
- Class library on top of ROOT
- Analysis controlled via interpreted or compiled C++ scripts (using ROOT's CINT interpreter)
- Toolbox of analysis modules
- Special emphasis on modularity
  - ▶ “Everything is a plug-in”
  - ▶ External user libraries dynamically loadable at run time
  - ▶ User code separate from core code
  - ▶ No need for users to write more than the code really needed (hopefully)
  - ▶ Core analyzer suitable for fixed installation, like ROOT itself
- Predefined modules for generic analysis tasks and standard Hall A equipment
- SDK available for rapid development of new module libraries

# Analysis Objects

- Any class that produces “results”
- Every analysis object has **unique name**, e.g. `R.s1`
- Results stored in “**global variables**”, prefixed with 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()`
  - ▶ `Init()`
  - ▶ etc.

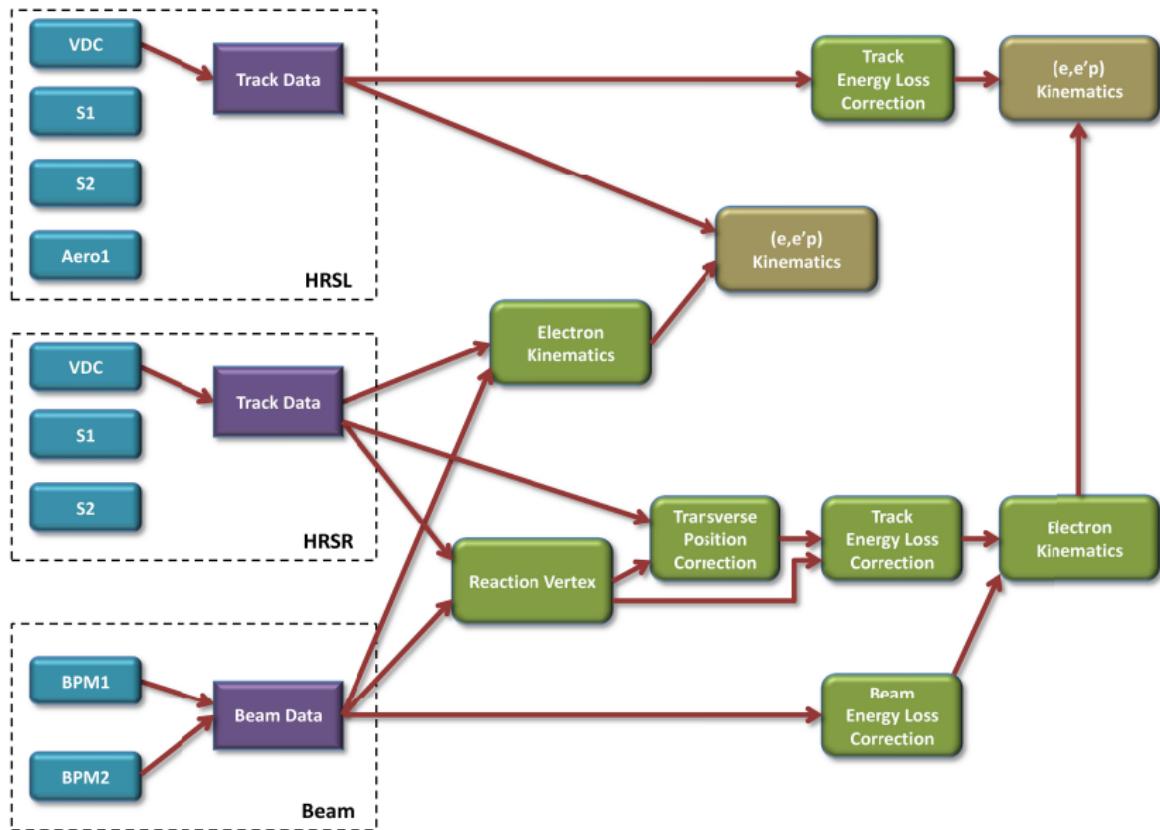
# Types of Analysis Objects

- “Detector”
  - ▶ Code/data for analyzing a **type** of detector.  
Examples: Scintillator, Cherenkov, VDC, BPM
  - ▶ Embedded in Apparatus or standalone
- “Apparatus” / “Spectrometer”
  - ▶ Collection of Detectors
  - ▶ Combines data from detectors
  - ▶ “**Spectrometer**”: Apparatus with support for **tracks** and standard `Reconstruct()` function
- “Physics Module”
  - ▶ Combines data from several apparatuses
  - ▶ Typical applications: **kinematics calculations, vertex finding, coincidence time extraction**
  - ▶ Special applications: debugging, event display
  - ▶ Toolbox design: Modules can be chained, combined, used as needed

# Tracks

- Ideally represent **4-vectors** of the reaction products
- THaTrack – reconstructed track with several sets of coordinates
- THaTrackInfo – generic TRANSPORT track coordinates
- Spectrometers store one or more THaTrack objects in a TConesArray
- **Multi-track capability** inherent in design
- Spectrometers may designate one of their tracks as the “**golden track**”
- Alternatively, decide on a golden track in a physics module, e.g. **THaGoldenTrack**

# C++ Analyzer Example Physics Module Chain



# C++ Analyzer User Interface

## Example Replay Script

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

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

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

// 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

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

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

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

// 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

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

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

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

// 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

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

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

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

// 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

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

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

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

// 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

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

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

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

// 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

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

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

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

// 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

```
# ---- Example e12345.odef -----  
  
# Variables to appear in the tree.  
variable L.s1.lt[4]  
variable L.s1.rt  
  
# The 'block' variables: All data in Right HRS go to tree.  
block R.*  
  
# Formulas can be scalers or vectors.  
# Lt4a is a scaler.  
formula Lt4a 5.*L.s1.lt[4]  
  
# Cuts can be defined globally and used in histograms.  
# Cut C1 is a scaler. Data is 0 or 1.  
cut C1 L.s1.lt[4]>1350  
  
# Histograms can involve formulas, variables, and cuts.  
# TH1F, TH1D, TH2F, TH2D supported.  
TH1F rvin 'L-arm vdc hits on V1' L.vdc.v1.nhit 10 0 10
```

# 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

```
# ---- Example e12345.odef -----  
  
# Variables to appear in the tree.  
variable L.s1.lt[4]  
variable L.s1.rt  
  
# The 'block' variables: All data in Right HRS go to tree.  
block R.*  
  
# Formulas can be scalers or vectors.  
# Lt4a is a scaler.  
formula Lt4a 5.*L.s1.lt[4]  
  
# Cuts can be defined globally and used in histograms.  
# Cut C1 is a scaler. Data is 0 or 1.  
cut C1 L.s1.lt[4]>1350  
  
# Histograms can involve formulas, variables, and cuts.  
# TH1F, TH1D, TH2F, TH2D supported.  
TH1F rv1n 'L-arm vdc hits on V1' L.vdc.v1.nhit 10 0 10
```

# 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 (CVS)
- Plan to investigate relational database system, e.g. Hall D's

## 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 (CVS)
- Plan to investigate relational database system, e.g. Hall D's

## 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](#)
- Allows terminating analysis of current event at various stages
- Inherits from TFormula → broad range of expressions supported
- Cut names available in output module as well

## 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](#)
- Allows terminating analysis of current event at various stages
- Inherits from TFormula → broad range of expressions supported
- Cut names available in output module as well

## 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
    EnoughShowrHits  R.sh.nhit>10
    Decode_master    !NoisyVDC
```

# Run objects

- **THaRun** (for CODA files on disk)
- Stores run info (file name, time, number, etc.)
- Stores global run parameters (beam energy, target parameters)
- Written to output file
- For **other input formats**, use/write class derived from **THaRunBase**

## Split Run Example

```
THaRun* r1 = new THaRun( "/data1/e01001_1000.dat.0" );
THaRun* r2 = new THaRun( "/data2/e01001_1000.dat.1" );
analyzer->Process( r1 );
analyzer->Process( r2 );
```

# Main Analyzer & Event Loop

- THaAnalyzer
- Manages various pieces of information:
  - ▶ Analysis objects
  - ▶ Output definitions
  - ▶ Cut definitions
  - ▶ Decoder
  - ▶ File names
  - ▶ Flags, etc.
- Coordinates initialization of all components
- Implements standard event loop [► doc](#)
- May write your own, even as a script

# Standard Spectrometer Processing

- ① For all tracking detectors
  - ▶ `CoarseTrack( tracks )`
  - ▶ Finds tracks without detailed, time-consuming corrections
- ② For all non-tracking detectors (e.g. PID detectors)
  - ▶ `CoarseProcess( tracks )`
  - ▶ Compute detector response, optionally using coarse tracks (read-only)
- ③ For all tracking detectors
  - ▶ `FineTrack( tracks )`
  - ▶ Repeat and/or refine tracking, optionally applying corrections and/or using `CoarseProcess` detector results
- ④ Reconstruct tracks to target
  - ▶ `FindVertices()`
- ⑤ For all non-tracking detectors
  - ▶ `FineProcess( tracks )`
  - ▶ (Re)compute detector response, optionally using fine tracks and/or target quantities
- ⑥ Compute additional attributes of tracks (e.g. momentum, beta, "Golden Track")
  - ▶ `TrackCalc()`
- ⑦ Combine all PID detectors to get overall PID for each track
  - ▶ `CalcPID()`

## Examples of Available Modules

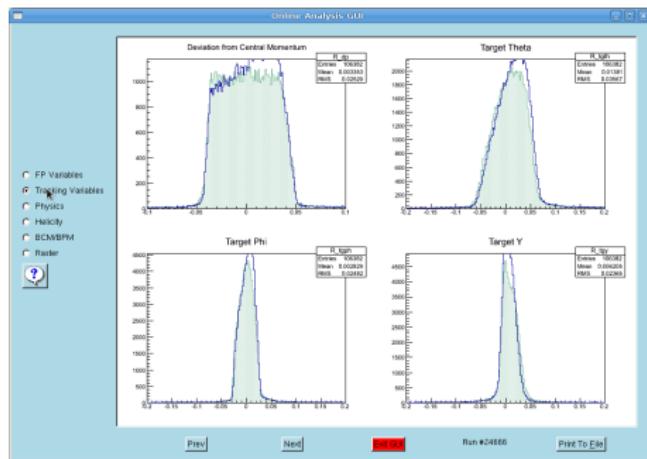
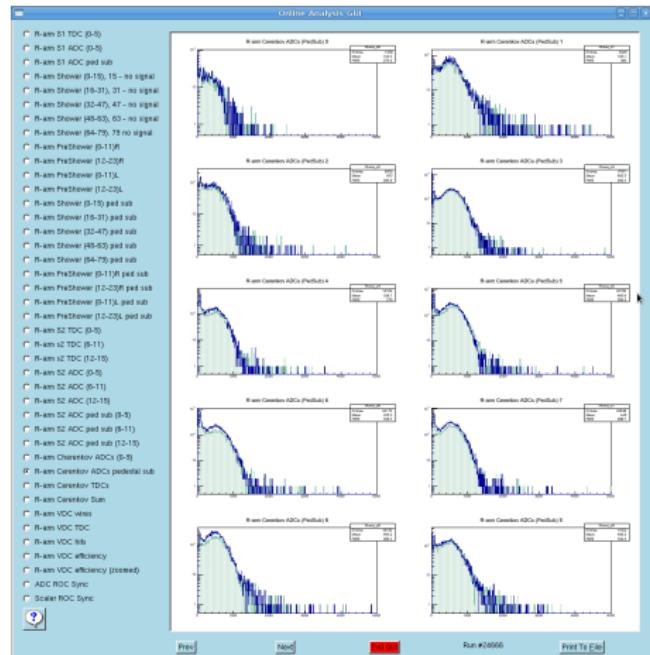
- CODA decoder (file & ET) with support for EPICS & scaler data
- Track reconstruction for HRS (VDCs) & BigBite (HDCs)
- Analysis of generic scintillators, Cherenkovs, shower counters
- BPM & raster analysis
- Vertex coordinate calculation
- Kinematics calculations for common reactions:  $(e, e')$ ,  $(e, e'X)$ ,  $(\gamma, X)$
- Coincidence time analysis
- Extended target ( $x_0$ ) corrections
- Energy loss corrections
- Dead-time calculation
- Helicity analysis (prompt & delayed modes)
- Decoding of generic hardware channels

# Online Histogramming

“OnlineGUI” (B. Moffit, 2007)

▶ pdf doc

▶ web



- Compiled ROOT script
- Visualizes DST (ROOT) output after prompt replay
- Easy configuration via text input file
- Reference histograms (shaded)

# C++ Analyzer Limitations

- Essentially **single-threaded**
  - ▶ Plan to implement multi-core support → this year
- Support for **12 GeV DAQ** environment not yet available
  - ▶ CODA 3 EVIO-library
  - ▶ Decoders for JLab 12 GeV **pipelined electronics**
- ROOT file **output performance** may be a bottleneck
  - ▶ Minor issue since ratio of reconstruction to output time is expected to rise with more demanding experiments
  - ▶ Mostly due to overbroad output definitions (too many variables)
- Smaller issues
  - ▶ Missing support for **pluggable extensions**: hardware module decoders (“drivers”), event types, track classes
  - ▶ **Documentation** not as good as it could be, especially for beginners
  - ▶ API and database format of C++ Analyzer classes sometimes inconsistent. Could benefit from **cleanup**.

# Development Plans

- Split off Hall A-specific code into separate library
- Tentatively: set up git repository, managed jointly with Hall C
- Implement multi-process architecture
- Add support for CODA 3 & pipelined DAQ front-ends
- Address remaining issues from previous slide as time permits

# Build System & Source Code Control

- Development environment
  - ▶ Standard GNU build tools (`make`, `g++` compiler)
  - ▶ Debugging similarly based on open source tools (`gdb`, `valgrind`)
  - ▶ No `autoconf` support (yet) → should add
- Source code revision control
  - ▶ C++ Analyzer under **CVS** since 2001
  - ▶ Full file history frequently useful
  - ▶ CVS may also provide text file database history (users' choice)
  - ▶ May migrate to git for compatibility with Hall C

# Documentation & Developer Support

- Web-based user guide [▶ go](#)
- THtml-generated reference documentation [▶ go](#)
- Example replay scripts (from previous experiments)
- Software development kit (SDK)
  - ▶ Facilitates rapid development of new apparatus, detector, and/or physics module classes. Provides skeleton code for each module type.
  - ▶ User code is placed in a shared library (plugin) that can be loaded dynamically at run time
  - ▶ No modifications of the core analyzer code necessary

# Software Development Kit (SDK) Examples I

## Example Physics Module Process() Function (simplified)

```
Int_t THaPrimaryKine::Process( const THaEvData& )
{
    // Calculate electron kinematics for the Golden Track of the given spectrometer

    // 4-momenta of incident & outgoing particle & target
    // NB: fP0 etc. are TLorentzVector objects

    fP0.SetVectM( fBeam->GetBeamInfo()->GetPvect(),      fM ); // e
    fP1.SetVectM( fSpectro->GetTrackInfo()->GetPvect(), fM ); // e'
    fA.SetXYZM( 0.0, 0.0, 0.0, fMA );                         // Target at rest

    // Textbook single-arm kinematics
    fQ        = fP0 - fP1;
    fQ2       = -fQ.M2();
    fQ3mag    = fQ.P();
    fOmega    = fQ.E();
    fA1       = fA + fQ;
    fW2       = fA1.M2();
    fTheta    = fP0.Angle( fP1.Vect() );
    fEpsilon  = 1.0 / ( 1.0 + 2.0*fQ3mag*fQ3mag/fQ2 *
                        TMath::Power( TMath::Tan(fTheta/2.0), 2.0 ) );
    fThetaQ   = fQ.Theta();
    fPhiQ    = fQ.Phi();

    fDataValid = true;
    return 0;
}
```

# Software Development Kit (SDK) Examples II

## 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 );
}
```

# Software Development Kit (SDK) Examples III

## Example ReadDatabase() Function (simplified)

```
Int_t THaADCHelicity::ReadDatabase( const TDatime& date )
{
    Int_t err = THaHelicityDet::ReadDatabase( date );
    if( err ) return kInitError; // these error checks omitted below

    FILE* file = OpenFile( date );

    vector<Int_t> heldef;
    fThreshold = kDefaultThreshold;
    Int_t ignore_gate = -1;
    const DBRequest request[] = {
        { "helchan",      &heldef,      kIntV,   0, 0, -2 },
        { "threshold",    &fThreshold,  kDouble, 0, 1, -2 },
        { "ignore_gate",  &ignore_gate, kInt,    0, 1, -2 },
        { 0 }
    };
    err = LoadDB( file, date, request, fPrefix );
    fclose(file);

    // Do the all-important consistency checks!
    if( heldef.size() != 3 ) {
        Error( Here(here), "Incorrect definition of helicity data channel. Must be "
              "exactly 3 numbers (roc,slot,chan), found %d. Fix database.", heldef.size() );
        return kInitError;
    }
    // ...
    fIsInit = true;
    return kOK;
}
```

# Summary

- Podd is mature software, in production use for almost 10 years
- Based on successful & widely adopted ROOT framework
- Very flexible design: maintainable & expandable
- Extensive experience & examples available