Hall A Analyzer Introduction

Ole Hansen

Jefferson Lab

Hall A & C Analysis Workshop
June 25, 2018

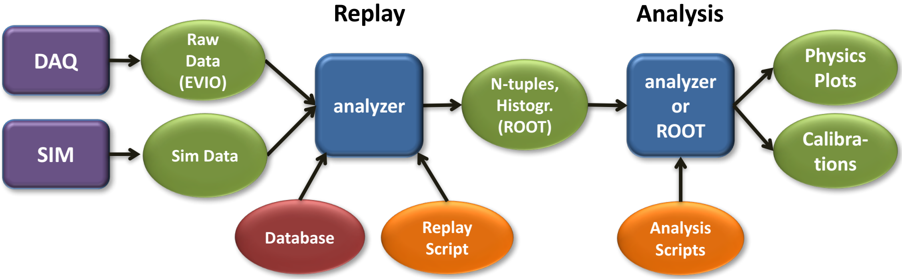
Prerequisites for doing analysis — Experimental Physics

- General nuclear physics, relativistic kinematics, detector principles
- Specific physics of your experiment
- Configuration of your experiment
  - Detector arrangement, geometry, DAQ/trigger
  - Run plan, run list, list of known issues
  - Resources: Experts, logbooks
- Good grasp of analysis techniques
  - Statistics, fitting, correlations
  - Cuts, conditions, run & event selection
  - Corrections for experimental effects
  - Particle identification techniques
  - Resources: (Textbooks), experts, analysis meetings, workshops like this

Prerequisites for doing analysis — Software Tools

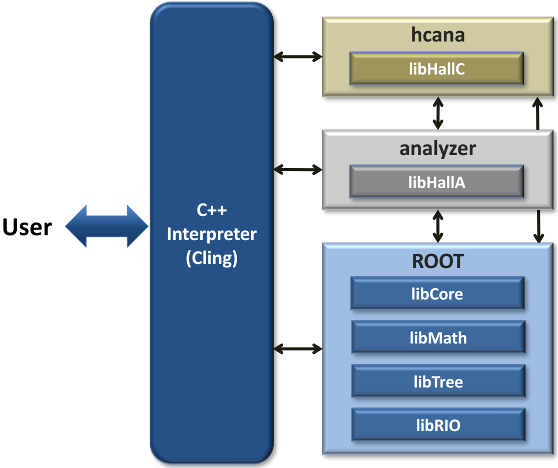
- Working knowledge of C++
  - Object-oriented programming (classes, polymorphism)
  - C++11 knowledge not essential at this time
  - Resources: Online tutorials, textbooks
- Familiarity with ROOT (although Python will often do too)
  - Resources: ROOT documentation (lots) `https://root.cern.ch/root-user-guides-and-manuals`
  - Good starting point: ROOT Primer
- Understanding of Hall A `analyzer` (and/or Hall C's `hcana`)
  - Basic concepts
  - Meaning of output variables
  - Resources: Documentation, workshops like this, experts, source code

Typical Hall A & C Analysis Flow



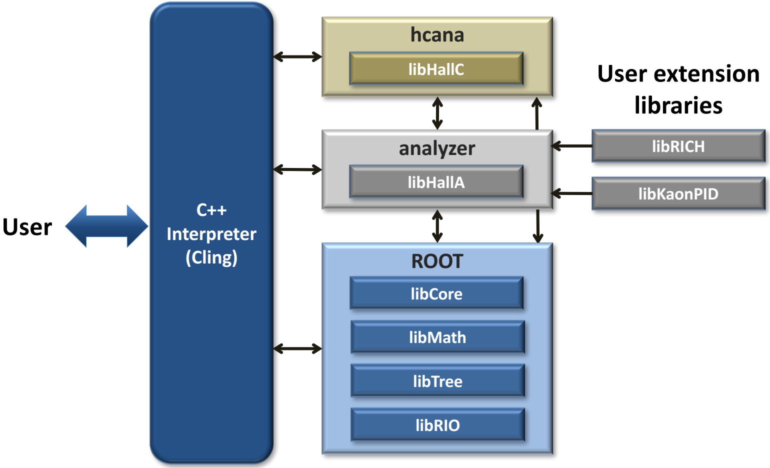
- **Replay:** Raw data → flat N-tuples in ROOT tree; histograms
- **Analysis:** ROOT files(s) → numerical results; plots
- Often necessary to run several replay and/or analysis passes

Analyzer as a ROOT Extension



- Hall A analyzer = Library of reconstruction & analysis classes on top of ROOT
- *All* of ROOT available at command line and programmatically

Plug-In Architecture



User

C++
Interpreter
(Cling)

hcana
libHallC

analyzer
libHallA

ROOT
libCore
libMath
libTree
libRIO

User extension libraries
libRICH
libKaonPID

SDK available

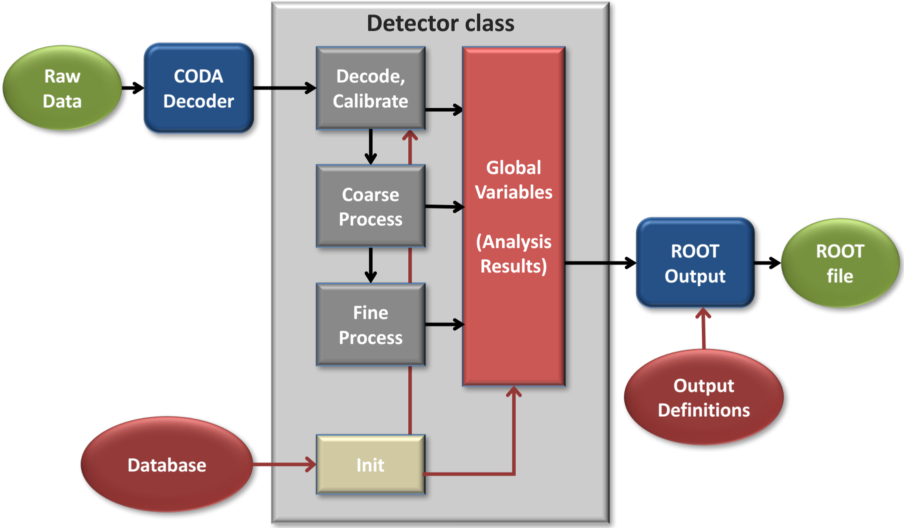
Analyzer Library: General Classes

- **Infrastructure**
  - Event loop (`THaAnalyzer`)
  - Database reader (`THaAnalysisObject::LoadDB`)
  - Raw data input interface (`THaEvData`)
  - ROOT output file writer (`THaOutput`)
- **Basic Reconstruction**
  - Standard detectors (*e.g.* `THaCherenkov`)
  - Spectrometer base class (`THaSpectrometer`)
  - Particle track data (`THaTrack`)
  - Incident beam (*e.g.* `THaUnrasteredBeam`)
- **Basic Analysis**
  - Kinematics calculations (*e.g.* `THaElectronKine`)
  - Vertex reconstruction (*e.g.* `THaReactionPoint`)
  - Energy loss calculation (*e.g.* `THeTrackEloss`)

Analyzer Library: JLab & Hall A-Specific Classes

- **JLab**
  - Raw data decoder (`Decoder::CodaDecoder`)
  - Rastered beam (`THaRasteredBeam`)
  - Beam helicity analysis (*e.g.* `THaADCHelicity`)
- **Hall A**
  - VDC track reconstruction (`THaVDC`)
  - HRS spectrometer (optics, target reconstruction) (`THaHRS`)
- **Hall C** has its own hall-specific library (see next talk)

Anatomy of a Detector Class



Class Categories

- **Detector**
    - Typically embedded in an Apparatus
    - Detectors should not know about each other (data encapsulation)
- **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
- Multiple instances of each type of object possible

"Global" Variables (Analysis Results)

- **Names of Analysis Object Instances**
  - Each *instance* of an Analysis Object has a unique name
  - Convention for detectors:

Object name = spectrometer name + "." + detector name

  - Example name: "R.cer": Right HRS ("R") gas Cherenkov ("cer")

- **"Global Variables"**
  - Give access to analysis results (stored in class member variables)
  - Can be a single value or fixed- or variable-size array
  - Available "globally" (in a global list: `gHaVars`)
  - Each variable has a unique name:

Variable name = Analysis Object Name + "." + Local Name

  - Example: "R.cer.asum_c" (Corrected ADC sum of "R.cer")

Database Files

Example Database File `~/Workshop2017/DB/20160205/db_R.cer.dat`

```
----[ 2016-02-05 00:00:00 -0500 ]
R.cer.detmap =
      1    20    32    41     1  1881
      2    11    32    41     1  1877
R.cer.npmt = 10
R.cer.position = 0  0  1.99
R.cer.size = 1  0.4  1
R.cer.tdc.offsets = 0  0  0  0  0  0  0  0  0  0
R.cer.adc.pedestals = 439.3  383.5  352.2  492.7  557.1  553  563.1  489.4  227.2  465.6
R.cer.adc.gains = 1.06  0.92  1.08  1.05  0.99  0.99  1  1.01  1.01  0.97

----[ 2016-09-10 00:00:00 -0400 ]
R.cer.position = -0.08  -0.008  1.8
R.cer.size = 1.22  0.302  1.37
R.cer.adc.pedestals = 439.8  384.3  352.8  493.1  557.1  553.2  564.1  490  227.3  465.9
R.cer.adc.gains = 0.926  0.919  1.139  1.002  0.95  0.997  0.989  1.014  1.05  0.983
```

- Flat text files of key/value pairs
- Values can be scalars, arrays, matrixes, strings
- Support for incremental validity periods and time zones
- Suitable for version control
- Currently must consult source code for list of recognized keys

Database Files

Example Database File `~/Workshop2017/DB/20160205/db_R.cer.dat`

```
----[ 2016-02-05 00:00:00 -0500 ]
R.cer.detmap =
      1    20    32    41      1  1881
      2    11    32    41      1  1877
R.cer.npmt = 10
R.cer.position = 0  0  1.99
R.cer.size = 1  0.4  1
R.cer.tdc.offsets = 0  0  0  0  0  0  0  0  0  0
R.cer.adc.pedestals = 439.3  383.5  352.2  492.7  557.1  553  563.1  489.4  227.2  465.6
R.cer.adc.gains = 1.06  0.92  1.08  1.05  0.99  0.99  1  1.01  1.01  0.97

----[ 2016-09-10 00:00:00 -0400 ]
R.cer.position = -0.08  -0.008  1.8
R.cer.size = 1.22  0.302  1.37
R.cer.adc.pedestals = 439.8  384.3  352.8  493.1  557.1  553.2  564.1  490  227.3  465.9
R.cer.adc.gains = 0.926  0.919  1.139  1.002  0.95  0.997  0.989  1.014  1.05  0.983
```

- Flat text files of key/value pairs
- Values can be scalars, arrays, matrixes, strings
- Support for incremental validity periods and time zones
- Suitable for version control
- Currently must consult source code for list of recognized keys

Dynamic Output Configuration

- Choose "global variables" to include in ROOT output tree
- No recompilation necessary

Example Output Definition File

```
# A single variable: Number of tracks found in the RHRS
variable R.tr.n
# A wildcard expression: all variables from the GoldenTrack module
block R.gold.*
# All RHRS track data (focal plane as well as at target)
# (usually too much information, narrow it down!)
block R.tr.*
```

- Much more possible
  - Arithmetic expressions
  - 1D and 2D histograms
  - Defining and appying cuts
  - Scalers
  - EPICS (slow control) variables
- Documentation: `https://redmine.jlab.org/projects/podd/wiki/Output`

Dynamic Output Configuration

- Choose "global variables" to include in ROOT output tree
- No recompilation necessary

Example Output Definition File

```
# A single variable: Number of tracks found in the RHRS
variable R.tr.n
# A wildcard expression: all variables from the GoldenTrack module
block R.gold.*
# All RHRS track data (focal plane as well as at target)
# (usually too much information, narrow it down!)
block R.tr.*
```

- Much more possible
  - Arithmetic expressions
  - 1D and 2D histograms
  - Defining and appying cuts
  - Scalers
  - EPICS (slow control) variables
- Documentation: `https://redmine.jlab.org/projects/podd/wiki/Output`

Unique capability, few other frameworks offer it!

Replay Example

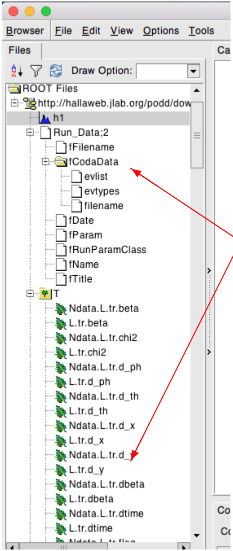
See last year's workshop:

https://redmine.jlab.org/projects/podd/wiki/Workshop2017/

Re-doing 2017 Example Replay in 2018 Virtual Machine

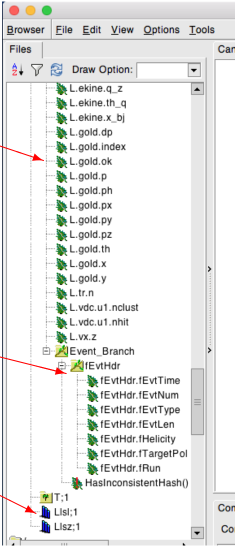
```
[wrkshp@centos7 ~]$ cd
[wrkshp@centos7 ~]$ git clone https://github.com/JeffersonLab/HallAC-
Workshop2017.git Workshop2017
[wrkshp@centos7 ~]$ source Workshop2017/setup.sh
[wrkshp@centos7 ~]$ cd Workshop2017/replay
[wrkshp@centos7 replay]$ analyzer
analyzer [0] .x replay.C
Here are the data files:
g2p_3132.dat.0
Run number? 3132
Number of events to replay (-1=all)? -1
...
314292  events read
204327  events accepted
Physics_master   GoodGoldenTrack      313476     203511 (64.9%)
...
analyzer [1] b = new TBrowser
```

Analyzer Output Structure



```
ROOT Files
  http://hallaweb.jlab.org/podd/dow
    h1
    Run_Data;2
      fFilename
      fCodaData
        evlist
        evtypes
        filename
      fDate
      fParam
      fRunParamClass
      fName
      fTitle
    T
      Ndata.L.tr.beta
      L.tr.beta
      Ndata.L.tr.chi2
      L.tr.chi2
      Ndata.L.tr.d_ph
      L.tr.d_ph
      Ndata.L.tr.d_th
      L.tr.d_th
      Ndata.L.tr.d_x
      L.tr.d_x
      Ndata.L.tr.d_y
      L.tr.d_y
      Ndata.L.tr.dbeta
      L.tr.dbeta
      Ndata.L.tr.dtime
      L.tr.dtime
```

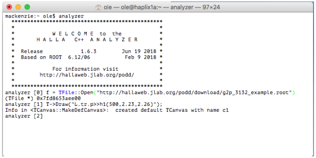
- Main results: tree leaves with basic data (scalars or arrays)
- Ndata variables are size counters for corresponding arrays
- Custom `analyzer` objects (metadata)
- Custom object hierarchy in tree
- Histograms saved in file



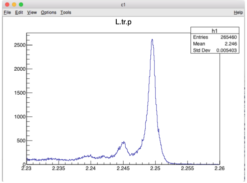
```
    L.ekine.q_z
    L.ekine.th_q
    L.ekine.x_bj
    L.gold.dp
    L.gold.index
    L.gold.ok
    L.gold.p
    L.gold.ph
    L.gold.px
    L.gold.py
    L.gold.pz
    L.gold.th
    L.gold.x
    L.gold.y
    L.tr.n
    L.vdc.u1.nclust
    L.vdc.u1.nhit
    L.vx.z
    Event_Branch
      fEvtHdr
        fEvtHdr.fEvtTime
        fEvtHdr.fEvtNum
        fEvtHdr.fEvtType
        fEvtHdr.fEvtLen
        fEvtHdr.fHelicity
        fEvtHdr.fTargetPol
        fEvtHdr.fRun
      HasInconsistentHash()
  T;1
  Llsl;1
  Llsz;1
```

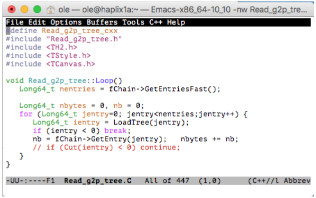
Analyzing Output ROOT Files

- Trees and histograms can be read by plain ROOT
- Reading run metadata & event headers requires `analyzer` or `hcana`
- Options
  - ROOT's `TBrowser` and `TTreeViewer`
  - Command line `T->Draw()`
  - Scripted/compiled custom loop
- Often must combine many ROOT files



```
mackenzie:~ ole$ analyzer
  ***********************************************
  *                                             *
  *            W E L C O M E  to  the           *
  *       H A L L A   C++  A N A L Y Z E R      *
  *                                             *
  *  Release           1.6.3        Jun 19 2018 *
  *  Based on ROOT  6.12/06          Feb 9 2018 *
  *                                             *
  *            For information visit            *
  *        http://hallaweb.jlab.org/podd/       *
  *                                             *
  ***********************************************
analyzer [0] f = TFile::Open("http://hallaweb.jlab.org/podd/download/g2p_3132_example.root")
(TFile *) 0x7fd8653aee00
analyzer [1] T->Draw("L.tr.p>>h1(500,2.23,2.26)");
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
analyzer [2]
```



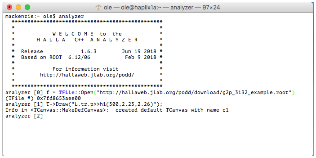

ole — ole@haplix1a:~ — Emacs-x86_64-10_10 -nw Read_g2p_tre...
File Edit Options Buffers Tools C++ Help

```
#define Read_g2p_tree_cxx
#include "Read_g2p_tree.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void Read_g2p_tree::Loop()
   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
}

-UU-:----F1  Read_g2p_tree.C   All of 447  (1,0)      (C++//l Abbrev
```

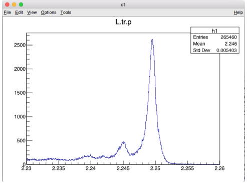
Analyzing Output ROOT Files

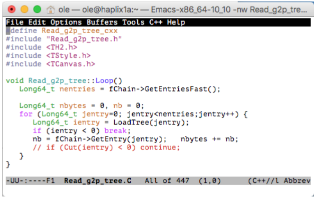
- Trees and histograms can be read by plain ROOT
- Reading run metadata & event headers requires `analyzer` or `hcana`
- Options
  - ROOT's `TBrowser` and `TTreeViewer`
  - Command line `T->Draw()`
  - Scripted/compiled custom loop
- Often must combine many ROOT files

> See tomorrow afternoon's talk on reading trees



```
mackenzie:~ ole$ analyzer
  ************************************************
  *                                              *
  *            W E L C O M E  to  the            *
  *       H A L L A   C++  A N A L Y Z E R       *
  *                                              *
  *  Release           1.6.3        Jun 19 2018 *
  *  Based on ROOT  6.12/06          Feb 9 2018 *
  *                                              *
  *            For information visit             *
  *        http://hallaweb.jlab.org/podd/        *
  *                                              *
  ************************************************
analyzer [0] f = TFile::Open("http://hallaweb.jlab.org/podd/download/g2p_3132_example.root")
(TFile *) 0x7fd8653aee00
analyzer [1] T->Draw("L.tr.p>>h1(500,2.23,2.26)");
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
analyzer [2]
```




ole — ole@haplix1a:~ — Emacs-x86_64-10_10 -nw Read_g2p_tre...
File Edit Options Buffers Tools C++ Help

```
#define Read_g2p_tree_cxx
#include "Read_g2p_tree.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void Read_g2p_tree::Loop()
   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
}

-UU-:----F1  Read_g2p_tree.C   All of 447  (1,0)      (C++//l Abbrev
```

Status & Outlook

- Version 1.6.0 was *finally* released on 3/14/2018 (Pi Day :))
  - New database format
  - Decoder modules
  - Improved VDC track reconstruction
  - Improved formula & test package (removed limitations)
  - See Release Notes for full list
- Current stable version: 1.6.3 (17-Jun-2018)
- New home page and issue tracker (Redmine)
  `https://redmine.jlab.org/projects/podd/wiki/`
- Development version: 1.7-devel (ETA 1.7.0 end of 2018)
  - Many improvements planned, see feature list on Redmine
    - Unified database interface
    - 3-parameter VDC cluster fits (needed for APEX)
    - More output options (non-`double` types, objects)
  - Work started

Resources

- Web site ▸ home page
    - Documentation
    - Release Notes
    - Source code downloads
    - Software Development Kit (included in distro)
    - Archived tutorials & example replays
- Issue & task tracker (Redmine) ▸ issues
- Mailing list: `halla_software@jlab.org`. Subscribe on ▸ mailman
- Analysis Workshop archive ▸ archive (includes older tutorials)