Introduction to the Hall A and Hall C Analysis Software

Ole Hansen

Jefferson Lab

Software & Computing Workshop
Jefferson Lab
May 15–19, 2023

Outline

External Libraries

Libraries import functionality beyond the core language. Example applications

- Convenience functions (*e.g.* string manipulation, containers)
- Text tokenizing & parsing
- Math, statistics, algorithms
- Database access
- Graphics, graphical user interfaces
- Multimedia
- Machine learning

Examples

- Boost (general purpose)
- `BLAS` (math)
- `Qt` (graphics)

Good list at `https://en.cppreference.com/w/cpp/links/libs`

C++ Library Types

Type	Pros	Cons
Header-only	– Support for templates – Portable: Included at compile time – Easy to include in project	– Slow compilation – Code bloat (memory, disk space) – Updates need recompilation
Static	– Portable: Included at link time	– Inefficient memory & disk use – Updates need recompilation
Shared/Dynamic	– Efficient resource use – Can be updated – Can be loaded dynamically (plugins)	– Must be installed on target systems – Version mismatches possible

- Header-only libraries are very popular in the C++ world
- Shared libraries are the de facto standard otherwise

Using External Libraries I

- If needed, install. Typically, need "development" package(s) (*-devel, *-dev)
- Include library header(s) in your code, similar to native C++ headers
- Unless header-only, explicitly link with library components

Example 1: Boost String Algorithms

```
#include <iostream>
#include <iomanip>  // for std::quoted (C++14)
#include <boost/algorithm/string.hpp>

using namespace std;
using namespace boost;

int main() {
   string s{" hello world! "};
   cout << quoted(s) << endl;
   to_upper(s);
   cout << quoted(s) << endl;
   trim(s);
   cout << quoted(s) << endl;
}
```

Build and run (NB: header-only)

```
$ sudo apt-get install libboost-dev
$ g++ -std=c++14 -o libex1 libex1.cc
$ ./libex1
" hello world! "
" HELLO WORLD! "
"HELLO WORLD!"
```

Using External Libraries II

Example 2: Readline

```
#include <iostream>
#include <iomanip>  // for std::quoted (C++14)
#include <cstdlib>  // for free()
#include <readline/readline.h>

using namespace std;

int main() {
   bool reading = true;
   while (reading) {
      char* line = readline("prompt> ");
      if (!line)
         reading = false;
      else {
         cout << "Read: " << quoted(line) << endl;
         free(line);
      }
   }
}
```

Build and run

```
$ sudo apt-get install libreadline-dev
$ g++ -std=c++14 -o libex2 libex2.cc -l readline
$ ./libex2
prompt>  hello there
Read: " hello there"
prompt> ^C
```

Building Libraries I (Linux)

- Linking with an existing library

```
$ g++ <options> <sources> -I /path/to/headers -L /path/to/lib -l libname
```

- Building and using your own shared library

```
$ g++ <options> -fPIC -c mylib.cc  # Builds mylib.o
$ g++ -shared -o libmylib.so mylib.o
$ g++ -o myprog myprog.cc -L. -l mylib
```

- Loading your own shared library

```
$ export LD_LIBRARY_PATH=.:"$LD_LIBRARY_PATH"
$ ./myprog
... # output goes here
```

- CMake quickly becomes your friend once a project employs multiple libraries

Building Libraries II (Linux)

- Building and using your own shared library (alternatives)

```
$ g++ -shared -fPIC -o libmylib.so mylib.cc     # Build and link in one step
$ g++ -o myprog myprog.cc -L. -l mylib -Wl,-rpath=‘pwd‘  # Set library path
```

- Programs with RPATH/RUNPATH set know where to look for libraries

```
$ ./myprog
... # output goes here

$ objdump -x myprog | grep -i r.*path
  RUNPATH              /home/ole/Develop/src/cppdemo
```

- CMake quickly becomes your friend once your project employs multiple libraries

ROOT: HEP's Premier Library Collection — And More

- ROOT = Framework for large scale data handling
- Features
  - efficient data storage, access and query (petabytes!)
  - advanced statistical analysis
  - scientific visualization
  - simulation support: detector geometry, event display
  - parallel query engine (PROOF)
  - interactive C++11 interpreter
  - . . . and many more
- Open Source. GNU LGPL. ∼2M lines of code.
- Supported by ∼10 staff, primarily at CERN and Fermilab
- Tens of thousands of users in high-energy and nuclear physics, and elsewhere (*e.g.* finance)

This talk covers the current version, ROOT 6

Large Collection of Libraries

Graphics & Visualization

- 1D, 2D and 3D histograms
- Large variety of 2D/3D plot types
- Can save graphics in many formats (*e.g.* PDF, JPEG, PNG)

Minimization

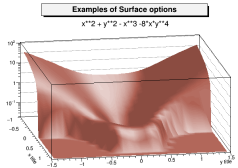
- Extensive library of fitting and minimization algorithms
- Widely used in data analysis and significance testing

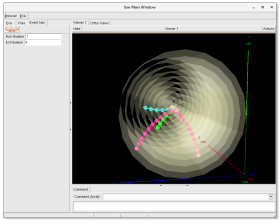
Mathematical Functions

- TMath: commonly used math functions, constants, statistics
- ROOT::Math (MathMore): very large collection of special functions, interface to GNU Scientific Library (GSL)

Geometry Toolkit and Event Display

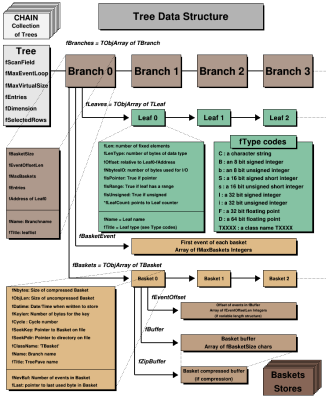
- Detailed description of detector geometry and materials
- Used extensively in simulations and event reconstruction
- 3D visualization of geometry, detector hits and tracks





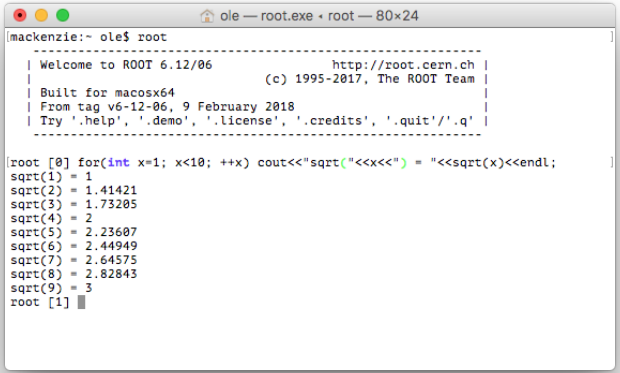
The Big Thing: C++ Object I/O and Persistency

- ROOT lets you write C++ objects to file ("serialization")
  - Extraordinary: impossible in native C++!
  - Achieved with C++ reflection, made possible by class dictionaries generated with the C++ interpreter Cling
- Can write single objects, collections (containers), entire object trees
  - Simple interface for all ROOT objects: `obj->Write()`
- Evolution from flat n-tuples used in low and medium-energy physics
- Cornerstone for the storage of experimental data in HEP. Used for storage of hundreds of petabytes at LHC
- Most relevant ROOT class: `TTree`
- Caveat: The recent trend in HEP is away from object storage in favor of flat files containing POD (plain old data) types (Python compatibility etc.). ROOT supports that too, of course ...



The Other Big Thing: The C++ Interpreter

- C++11 interpreter: `Cling` (based on LLVM/Clang)
- Interactive interface to all of ROOT
- Just-in-time compilation
- Allows rapid prototyping and testing
- Same language for scripting and compiled code



ROOT Setup/Installation

- Use the Hall A/C installation at JLab (should work on any RHEL 7 node that can see /group/halla)

```
$ ssh jlabl4
jlabl5.jlab.org[1] module use /group/halla/modulefiles
jlabl5.jlab.org[2] module avail
------------------------ /group/halla/modulefiles -------------------------
analyzer/1.7.0          evio/5.3(default)    group.apps           python/3.11.0        root/6.26.08(default)
analyzer/1.7.4(default) evio/5.3_gcc48       hcana/0.96           root/6.22.06         sbs-offline/20211206
analyzer/1.7.4_dbg      gcc/12.2.0           panguin/20211124     root/6.26.06
...

jlabl5.jlab.org[3] module load root
jlabl5.jlab.org[4] root
 ------------------------------------------------------------------
| Welcome to ROOT 6.26/08                        https://root.cern |
| (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for linuxx8664gcc on Nov 06 2022, 22:37:00                 |
| From heads/v6-26-08-halla@v6-26-08-4-g49482bde                   |
| With g++ (GCC) 12.2.0                                            |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
 ------------------------------------------------------------------

root [0] .demo
```

- See if ROOT is already installed at your home institution
- Follow the installation guide at `https://root.cern/install/`

Exercise 1: ROOT Command Line Basics

- May issue any C++11 statement, plus a few special ones (see later)
- May omit trailing ";" → return value printed
- May omit type specification → treated like "`auto`"
- Typing just the name of a defined variable prints it

Defining variables

```
$ root -l   # start up without logo (banner/splash screen)
root [0] int i = 10
(int) 10
root [1] float e = 2.718;
root [2] e
(float) 2.71800f
root [3] pi = 3.141592653
(double) 3.1415927
```

Exercise 2: ROOT Command Line Arithmetic

ROOT as a (hyper-charged) pocket calculator

```
root [4] 5+7*(3-2)
(int) 12
root [5] sqrt(2)
(double) 1.4142136
root [6] double angle = 45.;
root [7] sin(angle * TMath::DegToRad())
(double) 0.70710678
root [8] TMath::Erf(1)
(Double_t) 0.84270079
root [9] cout << setprecision(16) << TMath::Pi() << endl;
3.141592653589793
```

Exercise 3: C++11 & STL in ROOT

C++11 in ROOT: a few simple examples

```
// Initialize a std::vector: initializer list
root [1] vector<double> dvars {3.45, 1.5, 9.91, 6.28, -2.718}
(std::vector<double> &) { 3.45000, 1.50000, 9.91000, 6.28000, -2.71800 }

// Much simpler looping over containers (vectors etc.)
root [2] for( auto x : dvars ) cout << x << ", "; cout << endl;
3.45, 1.5, 9.91, 6.28, -2.718,

// STL algorithms
root [3] std::sort(dvars.begin(), dvars.end());
root [4] dvars
(std::vector<double> &) { -2.71800, 1.50000, 3.45000, 6.28000, 9.91000 }

// Lambda functions
root [5] std::sort(dvars.begin(), dvars.end(), [](double a, double b) { return b<a; } );

root [6] for( auto x : dvars ) cout << x << ", "; cout << endl;
9.91, 6.28, 3.45, 1.5, -2.718,
```

Exercise 4: Macros/Scripts

add42.C

```
int add42(int value) {
  return value+42;
}
int add53(int value) {
  return value+53;
}
```

Running and Loading a Macro/Script

```
// Run plain script. Runs function with same name as the file
root [1] .x add42.C(11)
(int) 53

// Load, then execute
// In this way, you can use more than one function defined in the script
root [2] .L add42.C
root [3] add42(11)
(int) 53
root [4] add53(12)
(int) 65
```

Exercise 5: Compiling Macros/Scripts

Compiling a macro/script

```
// Compile on the fly -- makes bigger scripts much faster
root [0] .L add42.C+
root [1] add42(11)
(int) 53

// Reuse compiled script -- rebuilt only if changed
// Execute directly
root [0] .x add42.C+(11)
(int) 53

// Load, then execute
root [0] .L add42.C+
root [1] add42(11)
(int) 53

// Force recompilation
root [2] .L add42.C++
```

Exercise 6: Opening and browsing ROOT files

ROOT files and the browser

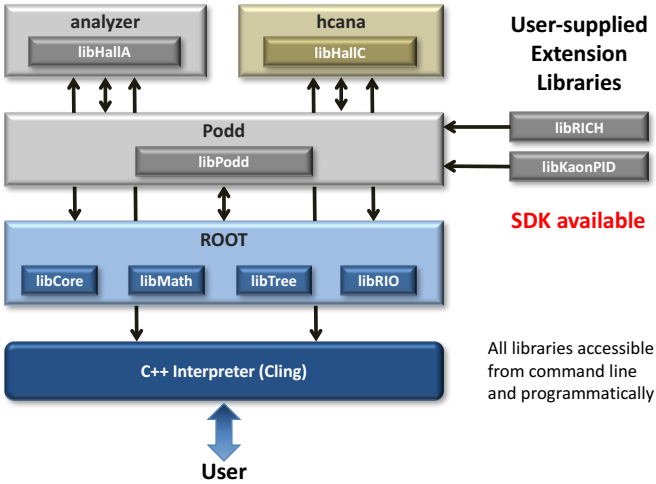
```
// Open a remote ROOT file via http (could also download first). These are actual Hall A data.
root [0]
 f = TFile::Open("http://hallaweb.jlab.org/podd/download/g2p_3132_example.root")
// (Ignore warnings about "no dictionary for class THa...")

// List the file contents. "KEY": on disk, "OBJ": in memory
root [1] .ls
TWebFile**      http://.../g2p_3132_example.root
 TWebFile*      http://.../g2p_3132_example.root
  KEY: THaRun   Run_Data;2     g2p run 3132 optics data
  KEY: TTree    T;1     Hall A Analyzer Output DST
  KEY: TH2F     Llsl;1  L u1 slope vs. local slope
  KEY: TH2F     Llsz;1  L u1 slope vs. cluster size

// Draw saved histogram:
root [2] Llsl->Draw("COLZ")

// Start up the user-friendly ROOT browser
root [3] new TBrowser;
```

The Hall A Analyzer ("Podd") — An Event Processing Framework

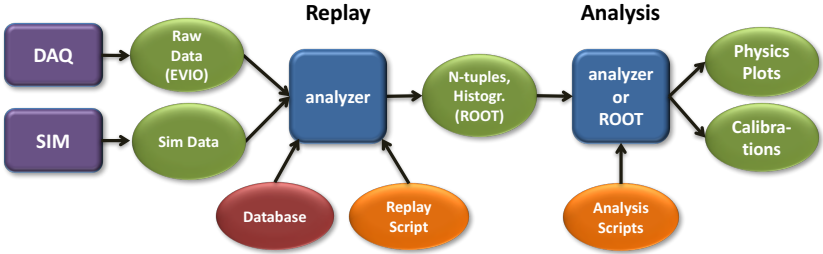
- C++ class library built on top of ROOT. Steering via ROOT interpreter.
- Developed in-house. Shared development with Hall C ("hcana").
- Documentation & bug tracker in Redmine. Sources on GitHub.
- Strengths
  - Highly modular to accommodate frequently changing experimental setups.
  - Intuitively conceptualizes analysis in terms of physical apparatuses (spectrometers, detectors) and physics calculations (kinematics, energy loss corrections, etc.)
  - Lightweight: minimal dependencies, small memory footprint.
  - Output & cuts run-time configurable via text files. Flat text file database.
- Limitations
  - Currently still single-threaded.
  - Designed for one-pass analysis: EVIO raw data $\rightarrow$ n-tuple-like ROOT trees + histograms
- Requirements
  - Linux or macOS
  - ROOT 6
  - CMake 3. C++11 compiler. (ROOT 6.26+ requires C++14.)

Modular Architecture

- User interface: ROOT prompt (C++ interpreter)
- All loaded libraries (ROOT, Podd, etc.) accessible from command prompt for scripting
- Extension libraries for experiment-specific code can be loaded dynamically
  - Software Development Kit (SDK) to get started
- Entire SBS software package implemented as such an extension



Reconstruction & Analysis Workflow



1. **Reconstruction (Replay)**
   - Runs in ROOT interpreter (`analyzer` prompt)
   - Calls mostly Podd functions & classes
   - Scripts set up by experiment experts or advanced users
   - After setup, runs in mass replay on the farm
2. **Analysis**
   - Also runs in ROOT interpreter (`analyzer` prompt)
   - Calls mostly ROOT functions and classes (but may need Podd classes)
   - Done by everyone on the experiment
   - Calibration and final physics usually done here

Podd Library: Core Classes

- **THaAnalysisObject**
  - Base class for all "analysis objects" (detectors, apparatuses, physics modules)
  - Provides standardized database access
  - Supports defining "analysis variables" (results)
- **THaAnalyzer**
  - Implements general-purpose event loop
  - Handles overall initialization
  - Offers convenience functions for setting file names etc.
- **THaRun & Podd::MultiFileRun**
  - Interface to input data files
  - Written to output file

All of these classes inherit from ROOT's TObject and so can be written to file

Types of Analysis Objects (inheriting from THaAnalysisObject)

- **Detector** (`THaDetector`)
  - Typically embedded in an Apparatus
  - Detectors should not know about each other (data encapsulation)
- **Apparatus / Spectrometer** (`THaApparatus` / `THaSpectrometer`)
  - Collection of Detectors
  - Combines data from detectors
  - "Spectrometer": Apparatus with support for tracks
- **Physics Module** (`THaPhysicsModule`)
  - 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

Examples of Actual Analysis Object Classes in Hall A Podd

- **THaSpectrometer**
  - `THaHRS`: Hall A High-Resolution Spectrometer (HRS)
- **THaApparatus**
  - `THaIdealBeam`: Static incoming particle beam
  - `THaRasteredBeam`: Dynamically moving ("rastered") incoming particle beam
- **THaDetector**
  - `THaScintillator`: Multi-paddle scintillator plane with two PMTs per paddle
  - `THaCherenkov`: Threshold Cherenkov counter with one or more PMTs
  - `THaShower`: Shower counter (calorimeter) with one or more blocks
  - `THaVDC`: Track reconstruction in multiple planes of horizontal drift chambers
- **THaPhysicsModule**
  - `THcHodoEff`: Hodoscope efficiency monitor
  - `THcBCMCurrent`: Beam current from BCM
  - `THcRFTime`: Accelerator RF time extraction

"Global" Variables (Analysis Results)

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

    Detector name = spectrometer name + "." + detector name

  - Example name: "R.s2": Right HRS ("R") scintillator 2 ("s2")
- **"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.s2.ra_c" (Corrected right-side PMT amplitudes of "R.s2" scintillator (array))

Setting Up a Replay (Reconstruction & Analysis Job)

- Tell Podd what detectors and spectrometers you are interested in, where to find data, where to put results, etc. → replay script
- Put databases together
  - Detector channels etc. ("cratemap", "detector maps", get these from your DAQ expert) → `db_cratemap.dat`
  - Geometry, calibration constants, flags → "db files"
- Tell Podd what results ("global variables") you want in the output file → output definition file ("odef")
- (Optional) Define tests (for statistics) and/or cuts (for event selection) → cut definition file ("cdef")
- Get the raw data files
- Find disk space for the output files, which can get large

Global Lists

Podd keeps global lists of analysis-related objects. They are accessible from the interpreter command line. You will need to interact with some of them.

`THaAnalyzer` takes these lists as input.

- `gHaApps`
  - Apparatuses (spectrometers, beam) that you want to analyze
  - You need to fill this list in your replay script before any real action can start
  - Apparatuses are processed in the order in which they appear here
  - The apparatuses contain the detectors, so there is no separate global list for them
- `gHaPhysics`
  - Physics Modules live here
  - Like `gHaApps`, fill this list with modules before starting any processing
- `gHaVars`
  - Home of the Global Variables. We've already covered them :)
  - You actually don't need to access this list directly very often

Example Replay Script

```
// Left-side HRS with VDC and S2 detectors
THaHRS* hrs = new THaHRS("L", "LHRS");
hrs->AddDetector( new THaVDC("vdc", "LHRS Vertical Drift Chambers"));
hrs->AddDetector( new THaScintillator("s2", "LHRS S2"));
hrs->AddDetector( new THaScintillator("s2alt", "LHRS S2 with alternate calibration"));
gHaApps->Add(hrs);

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

// Raw data input file
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile( "myresults.root" );
analyzer->SetOdefFile( "myreplay.odef" );

// Process all events in the input
analyzer->Process(run);
```

Example Replay Script

```
// Left-side HRS with VDC and S2 detectors
THaHRS* hrs = new THaHRS("L", "LHRS");
hrs->AddDetector( new THaVDC("vdc", "LHRS Vertical Drift Chambers"));
hrs->AddDetector( new THaScintillator("s2", "LHRS S2"));
hrs->AddDetector( new THaScintillator("s2alt", "LHRS S2 with alternate calibration"));
gHaApps->Add(hrs);

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

// Raw data input file
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOutFile( "myresults.root" );
analyzer->SetOdefFile( "myreplay.odef" );

// Process all events in the input
analyzer->Process(run);
```

Dynamic Output Configuration

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

Example Output Definition File (*e.g.* `myreplay.odef`)

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

User-Configurable N-tuple Output

Module header file:

```
  Int_t          GetNHits()          const { return fHits->GetLast()+1; }

protected:
  TClonesArray*  fWires;     // Wires
  TClonesArray*  fHits;      // Fired wires
  TClonesArray*  fClusters;  // Clusters

  Int_t fNHits;      // Total number of hits (including multihits)
  Int_t fNWiresHit;  // Number of wires with one or more hits
```

Module `DefineVariables()` method:

```
RVarDef vars[] = {
   { "nhit",   "Number of hits",                "GetNHits()" },
   { "wire",   "Active wire numbers",           "fHits.THaVDCHit.GetWireNum()" },
   { "rawtime","Raw TDC values of wires",       "fHits.THaVDCHit.fRawTime" },
   { "time",   "TDC values of active wires",    "fHits.THaVDCHit.fTime" },
   { "dist",   "Drift distances",               "fHits.THaVDCHit.fDist" },
   { "ddist",  "Drft dist uncertainty",         "fHits.THaVDCHit.fdDist" },
   { "trdist", "Dist. from track",              "fHits.THaVDCHit.ftrDist" },
   { "ltrdist","Dist. from local track",        "fHits.THaVDCHit.fltrDist" },
   { "trknum", "Track number (0=unused)",       "fHits.THaVDCHit.fTrkNum" },
   { "clsnum", "Cluster number (-1=unused)",    "fHits.THaVDCHit.fClsNum" },
   { "nclust", "Number of clusters",            "GetNClusters()" },
   { "clsiz",  "Cluster sizes",                 "fClusters.THaVDCCluster.GetSize()" },
   { "clspivot","Cluster pivot wire num",       "fClusters.THaVDCCluster.GetPivotWireNum()" },
   { "clpos",  "Cluster intercepts (m)",        "fClusters.THaVDCCluster.fInt" },
   { "slope",  "Cluster best slope",            "fClusters.THaVDCCluster.fSlope" },
```

Output definition file:

```
# All RHRS VDC raw data (lots of information, only uncomment for debugging)
# block ${arm}.vdc.*
block ${arm}.vdc.u1.*
```

ROOT file `TTree`:

```
*Br   17 :L.vdc.u1.wire : data[Ndata.L.vdc.u1.wire]/D                    *
*Entries :   265460 : Total  Size=   11970559 bytes  File Size  =    2295168 *
*Baskets :       81 : Basket Size=    2683904 bytes  Compression=   5.21     *
*Br   18 :L.vdc.u1.nclust : L.vdc.u1.nclust/D                            *
*Entries :   265460 : Total  Size=    2136991 bytes  File Size  =      45063 *
*Baskets :      131 : Basket Size=      64512 bytes  Compression=  47.36     *
*Br   19 :L.vdc.u1.nhit : L.vdc.u1.nhit/D                                *
*Entries :   265460 : Total  Size=    2136721 bytes  File Size  =     184378 *
*Baskets :      131 : Basket Size=      64512 bytes  Compression=  11.57     *
analyzer [11] T->Print("L.vdc.*")
```

Database Files

Example Database File $DB_DIR/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 $DB_DIR/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 Example (`ReadDatabase` method)

Validity timestamp ⟶

Scalar key/value pairs ⟶

Array key/values (auto-sized) ⟶

```
--------[ 2016-02-05 00:00:00 -0500 ]

R.vdc.nwires = 368
R.vdc.wire.spacing = -0.0042426
R.vdc.tdc.min = 800
R.vdc.tdc.max = 2200
R.vdc.tdc.res = 5e-10
R.vdc.ttd.param =
    2.12e-03   0.00e+00   0.00e+00   0.00e+00
   -4.20e-04   1.38e-03   1.06e-04   0.00e+00
    4.00e-09
R.vdc.t0.res = 6e-08
R.vdc.clust.minsize = 4
R.vdc.clust.maxspan = 7
R.vdc.maxgap = 0
R.vdc.tdiff.min = 3e-08
R.vdc.tdiff.max = 1.5e-07
R.vdc.u1.detmap =
     2    7    0   95    0
     2    8    0   95   96
     2    9    0   95  192
     2   10    0   79  288
R.vdc.u1.position = 0  0  0
R.vdc.u1.wire.start = 0.77852
R.vdc.u1.wire.angle = -45
R.vdc.u1.driftvel = 50000
R.vdc.u1.tdc.offsets =
    1533.3  1533.3  1533.3  1533.3  1533.3  1533.3  1533.3  1533.3  1533.3
    1533.3  1533.3  1533.3  1533.3  1533.3  1533.3  1533.3  1533.3  1533.3
    1534.1  1534.1  1534.1  1534.1  1534.1  1534.1  1534.1  1534.1  1534.1
    1534.1  1534.1  1534.1  1534.1  1534.1  1534.1  1534.1  1534.1  1534.1
    1534.7  1534.7  1534.7  1534.7  1534.7  1534.7  1534.7  1534.7  1534.7
    1534.7  1534.7  1534.7  1534.7  1534.7  1534.7  1534.7  1534.7  1534.7
    1533.9  1533.9  1533.9  1533.9  1533.9  1533.9  1533.9  1533.9  1533.9
    1533.9  1533.9  1533.9  1533.9  1533.9  1533.9  1533.9  1533.9  1533.9
    1534.2  1534.2  1534.2  1534.2  1534.2  1534.2  1534.2  1534.2  1534.2
    1534.2  1534.2  1534.2  1534.2  1534.2  1534.2  1534.2  1534.2  1534.2
    1526.4  1526.4  1526.4  1526.4  1526.4  1526.4  1526.4  1526.4  1526.4
    1526.4  1526.4  1526.4  1526.4  1526.4  1526.4  1526.4  1526.4  1526.4
    1533.8  1533.8  1533.8  1533.8  1533.8  1533.8  1533.8  1533.8  1533.8
    1533.8  1533.8  1533.8  1533.8  1533.8  1533.8  1533.8  1533.8  1533.8
    1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2
    1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2
    1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2
    1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2  1527.2
```

Retrieve parameter blocks in single call:

```
DBRequest request[] = {
  { "detmap",         &detmap,         kIntV },
  { "nwires",         &fNelem,         kInt,    0, 0, -1 },
  { "wire.start",     &fWBeg,          kDouble },
  { "wire.spacing",   &fWSpac,         kDouble, 0, 0, -1 },
  { "wire.angle",     &fWAngle,        kDouble, 0, 0 },
  { "wire.badlist",   &bad_wirelist,   kIntV,   0, 1 },
  { "driftvel",       &fDriftVel,      kDouble, 0, 0, -1 },
  { "tdc.min",        &fMinTime,       kInt,    0, 1, -1 },
  { "tdc.max",        &fMaxTime,       kInt,    0, 1, -1 },
  { "tdc.res",        &fTDCRes,        kDouble, 0, 0, -1 },
  { "tdc.offsets",    &tdc_offsets,    kFloatV },
  //...
  { 0 }
};
err = LoadDB( file, date, request, fPrefix );
```

More on Database Files

There are two special database files:

- `db_cratemap.dat`
  - Mapping of hardware locations (crate/slot) to electronics module types
  - Get this from your experiment's DAQ expert
  - Things will go very wrong if this file is incorrect
- `db_run.dat`
  - Run database. Contains time-dependent parameters about the experimental conditions (beam energy, spectrometer settings, etc.)
  - Required to be present with minimum set of parameters. If the values are off, so will be any kinematics calculated.

Additional things to keep in mind:

- Database file names normally have the form db_<analysis-object-name>.dat. Typically, only detectors and some physics modules require databases, although any analysis object can have one.
- Each detector database file needs to include a detector map (detmap), which specifies in which data acquisition hardware channels (crate/slot/channel) the data for this detector can be found.
- Set the environment variable $DB_DIR to the location of your database files.

Podd Setup/Installation

The easy way: Use Hall A/C installation on central systems

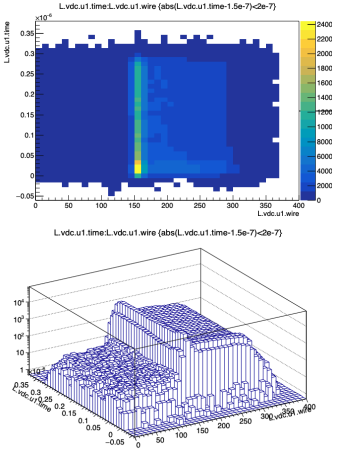
```
$ ssh jlab14
$ module use /group/halla/modulefiles
$ module load analyzer
$ module list
Currently Loaded Modulefiles:
1) group.apps   3) python/3.11.0          5) evio/5.3(default)
2) gcc/12.2.0   4) root/6.26.08(default)  6) analyzer/1.7.4(default)
$ analyzer --version
Podd 1.7.4 git@Release-174-0-ga0613dca 6 Nov 2022
Built for CentOS-7 using gcc-12.2.0, ROOT 6.26/08
$ 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            1.7.4         6 Nov 2022 *
*  Based on ROOT    6.26/08        18 Oct 2022 *
*                                              *
*            For information visit             *
* https://redmine.jlab.org/projects/podd/wiki/ *
*                                              *
************************************************
analyzer [0] TMath::Sqrt(2)
(double) 1.4142136
```

General installation instructions: `https://redmine.jlab.org/projects/podd/wiki/Installation`

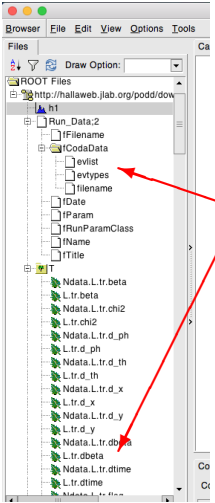
Using Podd/ROOT for Analysis — Mini Example

Mini example of tree visualization / "analysis"

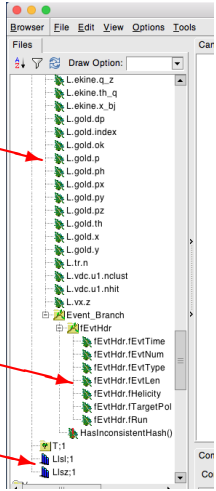
```
analyzer [1] f = TFile::Open("http://hallaweb.jlab.org/podd/download/g2p_3132_example.root");
analyzer [2] .ls
...
analyzer [3] Run_Data->Print();
THaRun  "RUN_3132"  "g2p run 3132 optics data"
Run number:   3132
Run date:     Mon Mar 12 19:21:44 2012
...
// Histogram of active VDC wires
analyzer [4] T->Draw("L.vdc.u1.wire")
// Override automatic binning: defining histograms on the fly
analyzer [5] T->Draw("L.vdc.u1.wire>>hw(368,0,368)","L.vdc.u1.wire>100")
// Applying a selection (reusing on-the-fly hw histogram)
analyzer [6] T->Draw("L.vdc.u1.wire>>hw","L.vdc.u1.wire>200")
// 2D histogram (2D plot) with selection and plot option
analyzer [7] T->Draw("L.vdc.u1.time:L.vdc.u1.wire","abs(L.vdc.u1.time-1.5e-7)<2e-7","COLZ")
// 2D histogram (3D plot) with selection and plot option
analyzer [8] T->Draw("L.vdc.u1.time:L.vdc.u1.wire","abs(L.vdc.u1.time-1.5e-7)<2e-7","LEGO")
// Set log-z scale (can also do this interactively by right-clicking near axis)
analyzer [9] gPad->SetLogz()

// Start ROOT browser for quick inspection of file/tree contents
analyzer [10] b = new TBrowser;
(see next page for browser details)
```



Browsing a ROOT File Generated by Podd



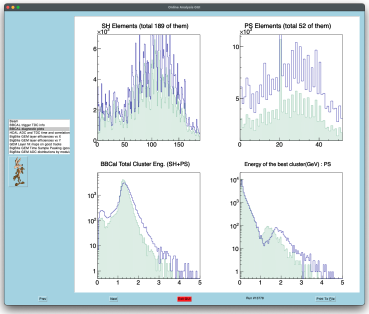
- Main results: tree leaves with basic data (scalars or arrays)
- Ndata variables are size counters for corresponding arrays
- Custom `analyzer` objects (metadata)
- Custom object hierarchy in tree
- Histograms saved in file
- Double-click on any leaf to draw a histogram of the variable



Panguin (Online GUI)

Panguin 2.5 Command Line Options

```
$ panguin --version
Panguin version 2.5 (23-Oct-2022)
$ panguin --help
panguin: configurable ROOT data visualization tool
Usage: panguin [OPTIONS]

Options:
-h,--help                    Print this help message and exit
-f,--config-file <file name> [default.cfg] Job configuration file
-r,--run <run number>        Run number
-R,--root-file <file name>   ROOT file to process
-G,--goldenroot-file <file name> Reference ROOT file
-P,-b,--batch                No GUI. Save plots to summary file(s)
-E,--plot-format <fmt>       Plot format (pdf, png, jpg ...)
-C,--config-dir <path>       Search path for configuration files & macros
(":"-separated)
--root-dir <path>            ROOT files search path (":"-separated)
-O,--plots-dir <dir>         Output directory for summary plots
-I,--images                  Save individual plots as images (implies -P)
-F,--image-format <fmt>      Image file format (png, jpg ...)
-H,--images-dir <dir>        Output directory for individual images
(default: plots-dir)
-v,--verbosity <level>       Set verbosity level (>=0)
-V,--version                 Display program version information and exit
```



- New command line options for easier scripting
- Configuration files support `include`
- File names and directory paths expand environment variables and placeholders:

  $ROOTFILES/$EXPERIMENT_%R.root

  summaryPlots_%R_page%P_%C.%E
- See README.md for full documentation

Adding New Classes

- Extensive documentation on how to add new detectors and apparatuses available
  https://redmine.jlab.org/projects/podd/wiki/Documentation
- Best place to start: **Software Development Kit** (included with analyzer).
  Documentation included: `README`, source code comments

Software Development Kit

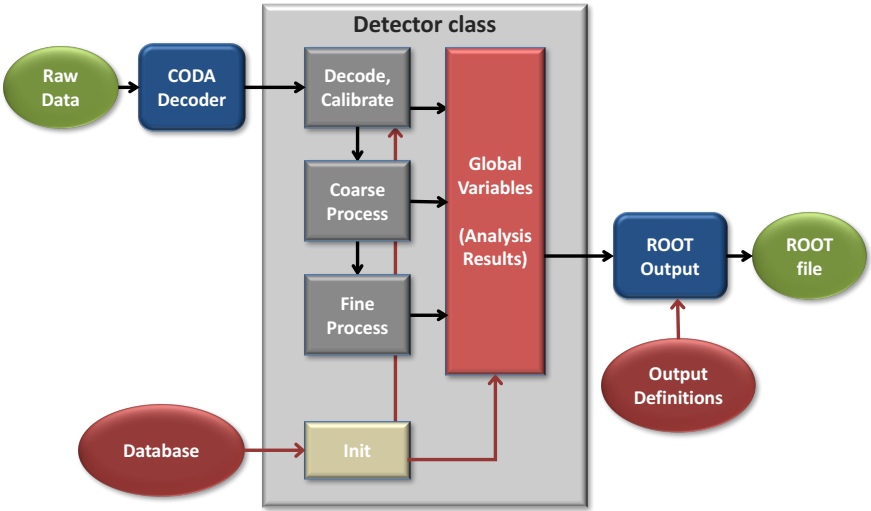
```
ifarm1401: ole$ ls $ANALYZER_SDK
CMakeLists.txt        UserApparatus.cxx     UserModule.h
Makefile              UserApparatus.h       UserScintillator.cxx
README.md             UserDetector.cxx      UserScintillator.h
SConscript.py         UserDetector.h        User_LinkDef.h
SConstruct            UserEvtHandler.cxx    config.log
SkeletonModule.cxx    UserEvtHandler.h      db_R.s1.dat
SkeletonModule.h      UserModule.cxx        db_U.u1.dat
```

Adding New Detectors: What Does a Detector Do?

Detectors provide the following functions to process each event

- All detectors
  - `Clear()`
  - Clears event-by-event data
  - `Decode( event_data )`
  - Retrieves raw data of interest from `event_data`
- "Tracking Detectors"
  - `CoarseTrack( tracks )`
  - Finds tracks without detailed, time-consuming corrections
  - `FineTrack( tracks )`
  - Repeat and/or refine tracking, optionally applying corrections and/or using `CoarseProcess` detector results
- "Non-Tracking Detectors"
  - `CoarseProcess( tracks )`
  - Compute detector response, optionally using coarse tracks (read-only)
  - `FineProcess( tracks )`
  - (Re)compute detector response, optionally using fine tracks and/or target quantities

Anatomy of a Detector Class



Example SDK Build

Example SDK Build (see also README.md)

```
$ module load cmake
$ cp -a $ANALYZER_SDK MyPoddSDK
$ cd MyPoddSDK
$ cmake -B build -S .
-- The CXX compiler identification is GNU 12.2.0
...
$ cmake --build build -j4
[ 11%] Generating UserDict.cxx, libUser_rdict.pcm
Scanning dependencies of target User
[ 22%] Building CXX object CMakeFiles/User.dir/UserDetector.cxx.o
...
[100%] Built target User
$
```

Example SDK Run

Example SDK Run

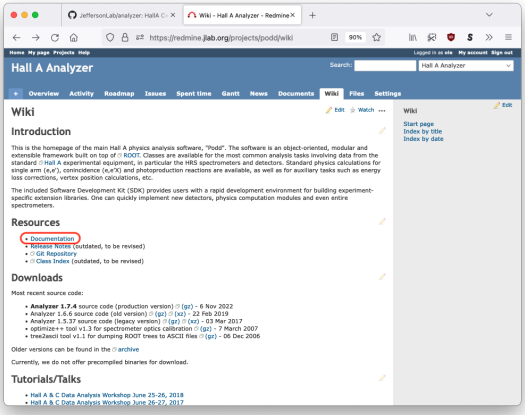
```
$ analyzer
analyzer [0] gSystem->Load("build/libUser")
(int) 0
analyzer [1] ua = new UserApparatus("U","User Apparatus");
analyzer [2] ua->AddDetector(new UserDetector("u1","User detector"));
Error in <UserApparatus::THaApparatus>: Detector with name u1 already
       exists for this apparatus. Not added.
analyzer [3] TDatime now;
analyzer [4] ua->Init(now)
(THaAnalysisObject::EStatus) (THaAnalysisObject::kOK) : (int) 0
analyzer [5] ua->Print("DETS")
AOBJ: UserApparatus U "U." User Apparatus
Collection name='TList', class='TList', size=1
AOBJ: UserDetector u1 "U.u1." User Detector 1
detmap = Size: 1
3    2    0    4    0    0   -1   -1       0    0    0
nelem = 5
pedestals = 15, 22, 14.5, 11.2, 17.6
gains = 3.3, 3.22, 3.54, 4.05, 3.42
nhits = 0
channel = (empty)
rawadc = (empty)
coradc = (empty)
```

Example SDK Run (cont.)

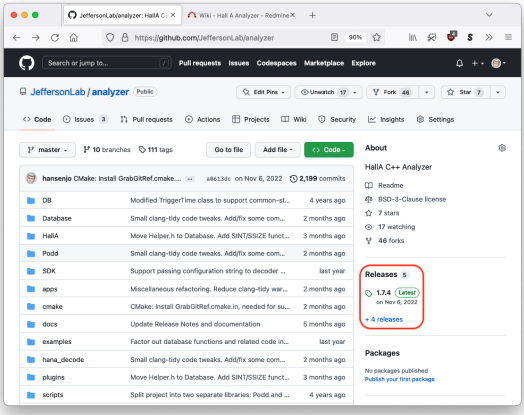
```
analyzer [6] gHaVars->Print("FULL")
Collection name='THaVarList', class='THaVarList', size=5
OBJ: THaVar U.ntot Total number of hits
(Int_t)    0
OBJ: THaVar U.u1.nhit Number of hits
(Int_t)    0
OBJ: THaVar U.u1.chan Channel number
(Int_t)  [0]
OBJ: THaVar U.u1.adc Raw ADC value
(Double_t)  [0]
OBJ: THaVar U.u1.adc_c Calibrated ADC
(Double_t)  [0]
analyzer [7] delete ua
analyzer [8] .q
```

Podd Documentation & Source Code

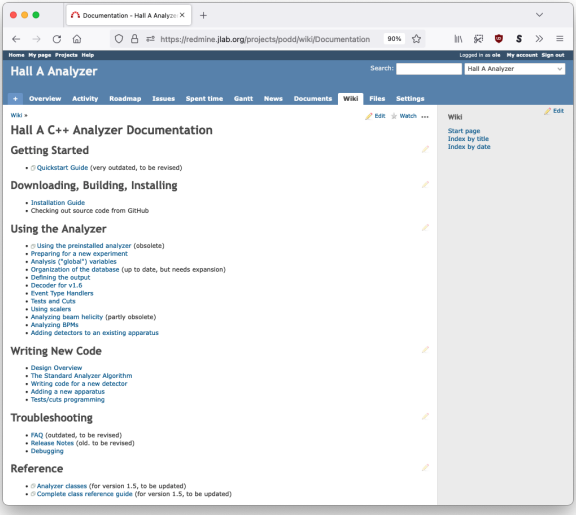
JLab Redmine



GitHub



Podd Documentation



- Some sections outdated/obsolete.
- Newer features not yet documented.
- A User's Guide and a formal publication would be nice.

Future Developments in Hall A

- Will be running the SBS (SuperBigBite Spectrometer) program through next year.
- The SBS collaboration has written an extensive custom software package for the special SBS detectors, which runs within Podd, processing petabytes of raw data.
- Starting in 2026, the MOLLER experiment will run in Hall A, which will bring its own, non-Podd software. Podd will still be used for occasional calibration measurements.
- Podd will likely be obsoleted in Hall A after MOLLER, except for legacy analyses.

The Hall C Analyzer ("hcana")

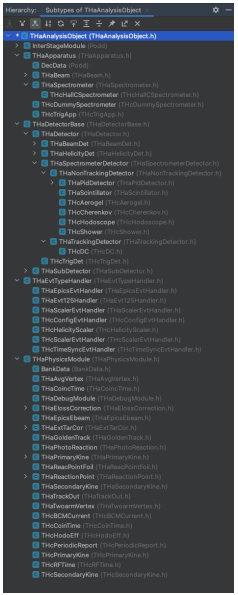
Main differences from Hall A analyzer

- Loads a separate library containing Hall C-specific classes (*e.g.* `THcHallCSpectrometer`) instead of the one with Hall A-only classes (which can be manually loaded, if needed).
- Uses different database system ("parameters"), a legacy of the old Hall C FORTRAN-based analysis code. Currently, this results in unnecessary duplication of many detector classes.
- Offers built-in support for end-of-run "reports".
- Detector decoders work differently by parsing so-called hit lists.
- Hall C replay scripts and parameter databases are neatly organized and maintained in a separate GitHub repository

Special Analysis Object Classes in `hcana`

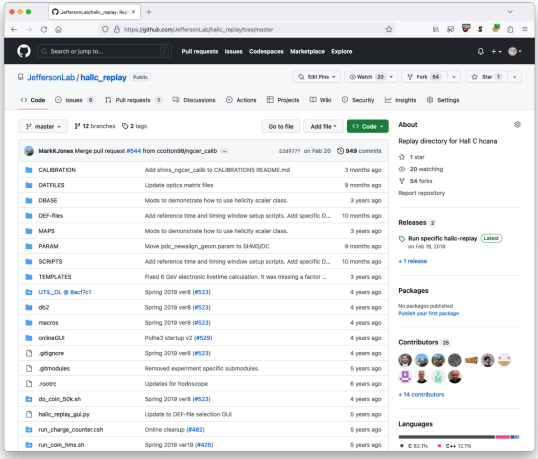
Classes that are substantially different or new compared to the Hall A analyzer

- **THaSpectrometer**
  - `THcHallCSpectrometer`: Generic Hall C Spectrometer (HMS or SHMS)
- **THaDetector**
  - `THcHodoscope`: Multiple planes of scintillators for coarse tracking and ToF
  - `THcAerogel`: Threshold aerogel Cherenkov counter with one or more PMTs
  - `THcDC`: Track reconstruction in vertical drift chambers (VDCs)
- **THaPhysicsModule**
  - `THaElectronKine`: Single-arm electron kinematics
  - `THaSecondaryKine`: Coincidence kinematics calculation (e,e'p)
  - `THaReactionPoint`: Vertex calculation for single-arm track
  - `THaTrackEloss`: Energy loss calculation for scattered particle

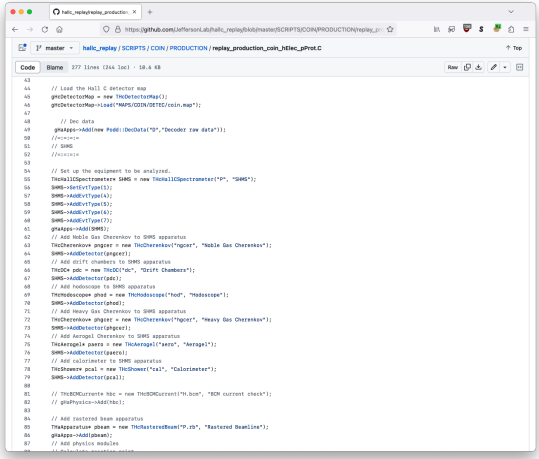


Hall C Replay Repository

GitHub

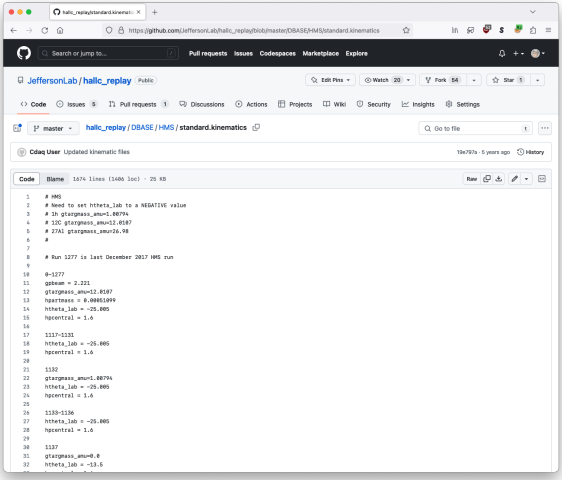


Replay Script Example



standard.kinematics file

hcana's equivalent to db_run.dat in Hall A: standard.kinematics

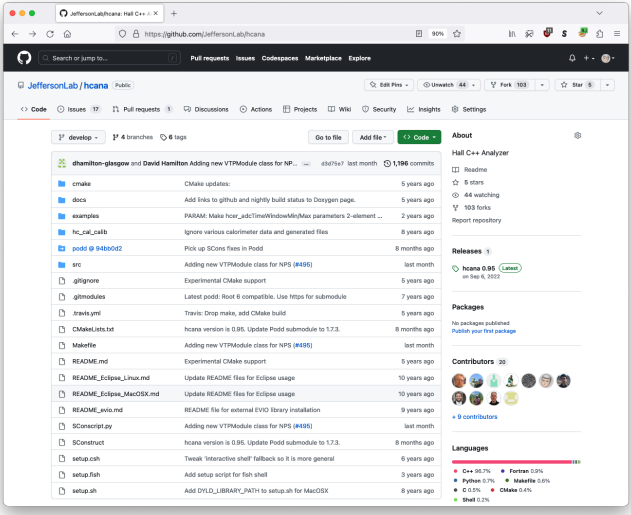


Using the Pre-Installed hcana

Getting hcana from the Hall A/C application area

```
$ module use /group/halla/modulefiles
$ module load hcana
$ hcana
**********************************************
*                                            *
*           W E L C O M E  to  the           *
*         H A L L C ++  A N A L Y Z E R      *
*                                            *
*  hcana release          0.96      07 Nov 2022 *
*  PODD release          1.7.4       6 Nov 2022 *
*  ROOT                6.26/08      Oct 18 2022 *
*                                            *
*           For information visit            *
*      http://hallcweb.jlab.org/hcana/docs/    *
*                                            *
**********************************************
hcana [0]
```

hcana Source Code



- See README for build instructions
- When checking out this repository, make sure to include the Podd git submodule (unless you already have Podd)
- (Some) documentation on Hall C Wiki

Future Developments in Hall C

- NPS (Neutral Particle Spectrometer) program will start this summer. NPS will have new software and use different DAQ readout mode. Both will be implemented in Podd/hcana.
- Longer-term program calls for mostly traditional HMS/SHMS experiments.
- `hcana` will likely be in use in its present form for the next 5+ years.

Thanks!