
# Hall C Analysis Software Upgrade - hcana

January 14, 2015

Stephen Wood

# Hall C Software Upgrade

Hall C used FORTRAN based analyzer "engine" for all non-parity experiments.
- Modernizing analysis software to remove dependence on Fortran and CERNLIB
- Chose to base new Hall C software on Hall A Podd framework.
  - ROOT well established and popular
    - (Hall C students often converted ntuples to root trees)
  - Will minimize cross training for users between Halls A/C

Imposing a distinct "Hall C" flavor to the code "hcana" we build on top of PODD.
- Replicating algorithms used in "engine"
  - Users trust those algorithms
  - Allows us to validate new code by comparing engine output to hcana
  - Detector classes written from scratch
- Using, to a partial extent, control files used by engine
  - Electronics channel to detector map file
  - Text based parameter/calibration files (CTP – CEBAF Test Package)
  - Hall A style DB files not used
  - Replace engine/CTP cut and histogram definition files with PODD equivalents

# Hall C Software Upgrade

Status

- Most detector code in place
  - Most detector classes will work for SHMS
  - New code needed for SHMS shower counter
  - Mods to detector classes for FADC250?
- Analysis results frequently compared to engine
- Needs work on “usability”
  - Calibration scripts (TOF, Optics, ...)
    - Only have for shower counter
  - Contents of root tree needs work
  - Scalers and EPICS event analysis needed
- Need to implement “nightly builds”

Code

- Available from https://github.com/JeffersonLab/hcana
- See readme.md for instructions to get started
- Questions: Zafar Ahmed, Ed Brash, Mark Jones, Gabriel Niculescu, Vardan Tadevosyan, Buddhini Waidyawansa, Stephen Wood, Simon Zhamkochyan

# ENGINE Parameter Files

ENGINE reads Calibration and Survey data from CTP parameter files.

New hcana can read these files so that
legacy FORTRAN analyzer and new ROOT analyzer use the same configuration files

```
;example.input
begin parm constants
  proton = 938.272   ; <- Comment character
  neutron = 939.57
  pion = 139.57
  pizero = 134.97
  offsets =
          0.34, 5.7, 0.1, -.4
          0.0, .1356, -1.3, 8.6
  nplanes = 6
  ratio = proton/neutron  ; Expressions allowed
end parm constants
```

# ThcParmList Class - 1

Extended Hall A analyzer ThaVarList class to read and parse CTP style parameter files.

```
../hcana
  ************************************************
  *                                              *
  *            W E L C O M E  to  the            *
  *       H A L L C   C++  A N A L Y Z E R       *
  *                                              *
  *        Release     1.5.25        Jan  9 2015 *
  *  Based on ROOT  5.34/21          Sep  9 2014 *
  *                                              *
  *            For information visit             *
  *        http://hallaweb.jlab.org/root/        *
  *                                              *
  ************************************************

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
hcana [0] gHcParms->Load("example.input")
```

# ThcParmList Class - 2

Extended Hall A analyzer ThaVarList class to read and parse CTP style parameter files.

```
hcana [1] gHcParms->PrintFull()
OBJ: THaVar  proton   <- Comment character
(Double_t)[1]  938.272
OBJ: THaVar  neutron
(Double_t)[1]  939.57
OBJ: THaVar  pion
(Double_t)[1]  139.57
OBJ: THaVar  pizero
(Double_t)[1]  134.97
OBJ: THaVar  offsets
(Double_t)[8]  0.34  5.7  0.1  -0.4  0  0.1356  -1.3  8.6
OBJ: THaVar  nplanes
(Int_t)[1]  6
OBJ: THaVar  ratio    Expressions allowed
(Double_t)[1]  0.998619

hcana [2] Double_t* offsets
hcana [3] offsets = gHcParms->Find("offsets")->GetValuePointer();
hcana [4] offsets[3]
(Double_t)(-4.0000000000000022e-01)
```

# ThcParmList Class - 3

Usage:

All paramters are read at start of analysis script

Detector classes request parameter values as needed

```
fPrefix[0] = tolower(GetApparatus()->GetName()[0]);

DBRequest list[]={
    {"dc_num_planes",&fNPlanes, kInt},
    {"dc_num_chambers",&fNChambers, kInt},
    {"dc_tdc_time_per_channel",&fNSperChan, kDouble},
    {"dc_wire_velocity",&fWireVelocity,kDouble},
    {"dc_plane_names",&planenamelist, kString},
    {0}
};

gHcParms->LoadParmValues((DBRequest*)&list,fPrefix);

DBRequest list[]={
     ...
    {"dc_xcenter", fXCenter, kDouble, fNChambers},
    {"dc_ycenter", fYCenter, kDouble, fNChambers},
    {"min_hit", fMinHits, kInt, fNChambers},
    {"max_pr_hits", fMaxHits, kInt, fNChambers},
    {"min_combos", fMinCombos, kInt, fNChambers},
    ...
     {0}
};

gHcParms->LoadParmValues((DBRequets*)&list,fPrefix);
```

# Experimental Hall C CCDB database

Experimental CCDB (Hall D Constants and Calibration Database, by Dmitry Romanov)
support has been added to hcana

https://halldweb1.jlab.org/wiki/images/a/a4/Ccdb_romanovda_02_11_2014.pptx

CCDB support hierarchical naming structure (directories)

Experimental CCDB has three directories

/hms
/sos
/gen

h*varname* -> /hms/*varname*
s*varname* -> /sos/*varname*
g*varname* -> /gen/*varname*

Examples

hpcentral -> /hms/pcentral
h_remove_sppt_if_one_y_plane -> /hms/_remove_sppt_if_one_y_plane

# Using CCDB in hcana

Define CCDB_CONNECTION to point to CCDB file:
e.g. Sqlite:////home/saw/ROOT/CCDB/hallc.sqlite

In hcana CINT script do:

```
gHcParms->OpenCCDB(RunNumber);
gHcParms->LoadCCDBirectory("hms","h")
gHcParms->LoadCCDBirectory("sos","s")
gHcParms->LoadCCDBirectory("gen","g")
```

This loads parameters into hcana and converts them back to original variable names.

No detector code needs to be modified.

TODO:

Develop scripts to convert set of PARAM and DBASE files into a CCDB that holds parameters with all their run number dependencies.
Develop GUIS to examine/modify parameters, particularly kinematic settings.

# Build hcana with CCDB

CCDB compatibility, by default, not compiled in hcana

1. Get latest version of ccdb (via download or svn from Hall D)
2. “source environment.bash” or “source environment.csh”
3. scons with-mysql=false with-examples=true
4. Build hcana with “make”  (scons support not available yet)

# Reports

engine included end of run printouts using the CTP report feature.
User defined text “template” with calculations using parameters, scalers, EPICS values, and test/cut statistics. Information from end of run reports often mined with perl scripts.

Report feature partially cloned in hcana

analyzer->PrintReport(filein, fileout);

Copies contents fo filein to fileout, “{stuff:fmt}” with the evaluated value of “stuff” formatted with the c-style format descriptor “fmt”.

Example:
Cut.def file contains “Pedestal_event g.evtype==4”

```
100*Pedestal_event.npassed/Pedestal_event.ncalled:%.2f} of the events are
   pedestal events
HMS reconstruction matrix filename is {h_recon_coeff_filename}

4.26% of the events are pedestal events
HMS reconstruction matrix filename is PARAM/hms_recon_coeff.dat
```

Expressions can involve parameters defined in gHcParms, or cuts defined in cut.def file. (Cuts in .odef file not supported)

Plan to add scalers and EPICS information.

(See examples/report.template for more examples)

# ENGINE Decoding

ENGINE decoder uses single map file to describe electronics channel to detector element mapping.

Sorts raw data into hit lists for each detector (Wire chamber set, hodoscope set, aerogel, shower counter ...). A hit consists of all the data for a given wire or bar.

```
1, Plane, counter, TDC          # WC hit list.  Plane, counter might be
2, Plane, counter, TDC          # repeated for multiple hits on one wire
...
```

For hodoscopes

```
1, Plane, counter, TDC+, TDC-, ADC+, ADC-
2, Plane, counter, TDC+, TDC-, ADC+, ADC-
...
```

In order to closely replicate algorithms, these hit lists are implemented in hcana.

# Raw Hit Lists

New Classes to build on top of Hall A analyzer decoder:

THcDetectorMap (.h, .cxx)– Reads and stores the Hall C style detector map. Global object gHcDetectorMap

THcRawHit – Abstract class that specific detector types will inherit their hits from. Must have at least plane and counter as members. Must have SetData method to save raw data values according to signal number (1-4 for hodoscopes)

THcAerogelHit, THcCherenkovHit, THcHodoscopeHit, THcRawDCHit, ThcRawShowerHit

Detector raw hit classes. Inherit from ThcRawHit

Have data members holding all raw data associated with a single counter (hodoscope bar, wire, lead glass block, ...)

e.g. For Hodoscope

fADC_pos, fADC_neg, fTDC_pos, fTDC_neg

THcRawHitList – A list of raw hits of of some hit type that inherits from THcRawHit. When given the decoded event object and the detector map object, it constructs the raw hit list for its detector

# Event Decoding - Usage

In driver script:

```
gHcDetectorMap = new THcDetectorMap;  //Should this be done in THcInterface?
gHcDetctorMap->Load(mapfilename);
```

In Detector class Init method:

```
InitHitList(fDetMap, rawhitlistclassname, maxhits);
InitHitList(fDetMap, "ThcRawHodoHit", 100);
```

Intializes TClonesArray of rawhit objects

```
gHcDetectorMap->FillMap(fDetMap, detectoridname);
gHcDetectorMap->FillMap(fDetMap, "HSCIN");
```

Creates the detector specific map in fDetMap, a ThaDetMap object.

In Detector class Decode method:

```
Int_t nhits = DecodeToHitList(evdata);
```

Puts data in fRawHistList, a TClonesArray of the detector specific raw hit type.

# db_cratemap.dat

Hall A analyzer requires `db_cratemap.dat` file.

```
   ==== Crate 1 type fastbus
# slot   model    clear    header   mask     nchan    ndata
  1      1881     1        0x0      0x0      64       64
...
 19      1875     1        0x0      0x0      64       64
...
==== Crate 2 type fastbus
# slot   model    clear    header   mask     nchan    ndata
  2      1877     1        0x0      0x0      64       256
...
```

Perl script `make_cratemap.pl` will create `db_cratemap.dat` from a Hall C mapfile. This currently must be done in driver script. May integrate into THcDetectorMap::Load?

# Modified ThaDetMap Class

```
struct Module {
    UShort_t crate;
    UShort_t slot;
    UShort_t lo;
    UShort_t hi;
    UInt_t   first;  // logical number of first channel
    UInt_t   model;  // model number of module (for ADC/TDC identification).
    //FIXME: Clean this up in next major release
#ifdef HALLC_MODS
    UInt_t   plane;  // Detector plane
    UInt_t   signal; // (eg. PosADC, NegADC, PosTDC, NegTDC)
#endif
    Int_t    refchan;    // for pipeline TDCs: reference channel number
    Int_t    refindex;   // for pipeline TDCs: index into reference channel
   map
    Double_t resolution; // Resolution (s/chan) for TDCs
    Bool_t   reverse;    // Indicates that "first" corresponds to hi, not lo
    ...
  };
```

# Example Analysis Script examples/hodtest.C - 1

Load run # dependent parameters

```
Int_t RunNumber=50017;
char* RunFileNamePattern="daq04_%d.log.0";

gHcParms->Define("gen_run_number", "Run Number", RunNumber);
gHcParms->AddString("g_ctp_database_filename", "DBASE/test.database");

gHcParms->Load(gHcParms->GetString("g_ctp_database_filename"), RunNumber);
// g_ctp_parm_filename and g_decode_map_filename should now be defined

gHcParms->Load(gHcParms->GetString("g_ctp_parm_filename"));
// Constants not in ENGINE PARAM files that we want to be configurable
gHcParms->Load("PARAM/hcana.param");


// Generate db_cratemap.dat to correspond to map file contents
char command[100];
sprintf(command,"./make_cratemap.pl < %s > db_cratemap.dat",gH
 >GetString("g_decode_map_filename"));
system(command);

// Load the Hall C style detector map
gHcDetectorMap=new THcDetectorMap();
gHcDetectorMap->Load(gHcParms->GetString("g_decode_map_filename"));
```

Load channel to detector mapping

# Example Analysis Script examples/hodtest.C - 2

```
// Set up the equipment to be analyzed.

THaApparatus* HMS = new THcHallCSpectrometer("H","HMS");
gHaApps->Add( HMS );

// Add hodoscope
HMS->AddDetector( new THcHodoscope("hod", "Hodoscope" ));
HMS->AddDetector( new THcShower("cal", "Shower" ));
HMS->AddDetector( new THcDC("dc", "Drift Chambers" ));
THcAerogel* aerogel = new THcAerogel("aero", "Aerogel Cerenkov" );
HMS->AddDetector( aerogel );
THcCherenkov* cherenkov = new THcCherenkov("cher", "Gas Cerenkov" );
HMS->AddDetector( cherenkov );

THaApparatus* SOS = new THcHallCSpectrometer("S","SOS");
gHaApps->Add( SOS );
// Add detectors
SOS->AddDetector( new THcHodoscope("hod", "Hodoscope" ));
SOS->AddDetector( new THcShower("cal", "Shower" ));
SOS->AddDetector( new THcDC("dc", "Drift Chambers" ));

ThcAnalyzer* analyzer = new THcAnalyzer;
ThaEvent* event = new ThaEvent;
```

# Example Analysis Script examples/hodtest.C - 3

```
// Define the run(s) that we want to analyze.
// We just set up one, but this could be many.
char RunFileName[100];
sprintf(RunFileName,RunFileNamePattern,RunNumber);
THaRun* run = new THaRun(RunFileName);

run->SetEventRange(1,100000);//  Physics Event number, does not
                              // include scaler or control events

// Define the analysis parameters
analyzer->SetEvent( event );
analyzer->SetOutFile( "hodtest.root" );
analyzer->SetOdefFile("output.def");
analyzer->SetCutFile("hodtest_cuts.def");        // optional
analyzer->SetCountMode(2);// Counter event number same as gen_event_ID_number

// File to record cuts accounting information
//  analyzer->SetSummaryFile("summary_example.log"); // optional

analyzer->Process(run);     // start the actual analysis

analyzer->PrintReport("report.template","report.out");
```

Generate end of run printout. Expressions involving cut success counts, parameters and eventually scalers.