Getting Started with the C++ Analyzer Tutorial

Ole Hansen

Jefferson Lab

Joint Hall A & C Data Analysis Workshop
December 12, 2012

Outline

Getting ROOT

- Use central installations at JLab
  - Counting house (adaq machines). Separate debug version installed for development.
  - CUE: `source /apps/root//PRO/bin/thisroot.csh`
- Package for your Linux distribution
  - Fedora 14+: `yum install root`. Must pick and choose subpackages, like `root-physics`.
  - Install separate debuginfo package(s) for development.
  - Ubuntu?
- Pre-compiled version from CERN
  - Make sure it matches your platform exactly!
  - Easy
  - Limited flexibility
  - No debug support (I think)
- Compile from source
  - Somewhat hard & error-prone
  - Use `do_configure` script on adaq as an example

Analyzer

- `wget http://hallaweb.jlab.org/podd/download/analyzer-1.5.24.tar.gz`
- Unpack
- Linux build: `make [-j3]`
- "Install": Set at least `LD_LIBRARY_PATH`
- A few additional steps necessary on MacOSX. See documentation.

Setting Up a Replay

- Determine detector configuration, desired calculatons, output variables needed, etc. ($\rightarrow$ demonstration)
- Write (or copy/modify) replay script, output definitions, optional cuts/tests
- Create (or copy/modify) database files. This can be time-consuming! Starting with approximate calibration data is almost always fine.
- Get raw data files
- Of course, do simple sanity checks before running large jobs

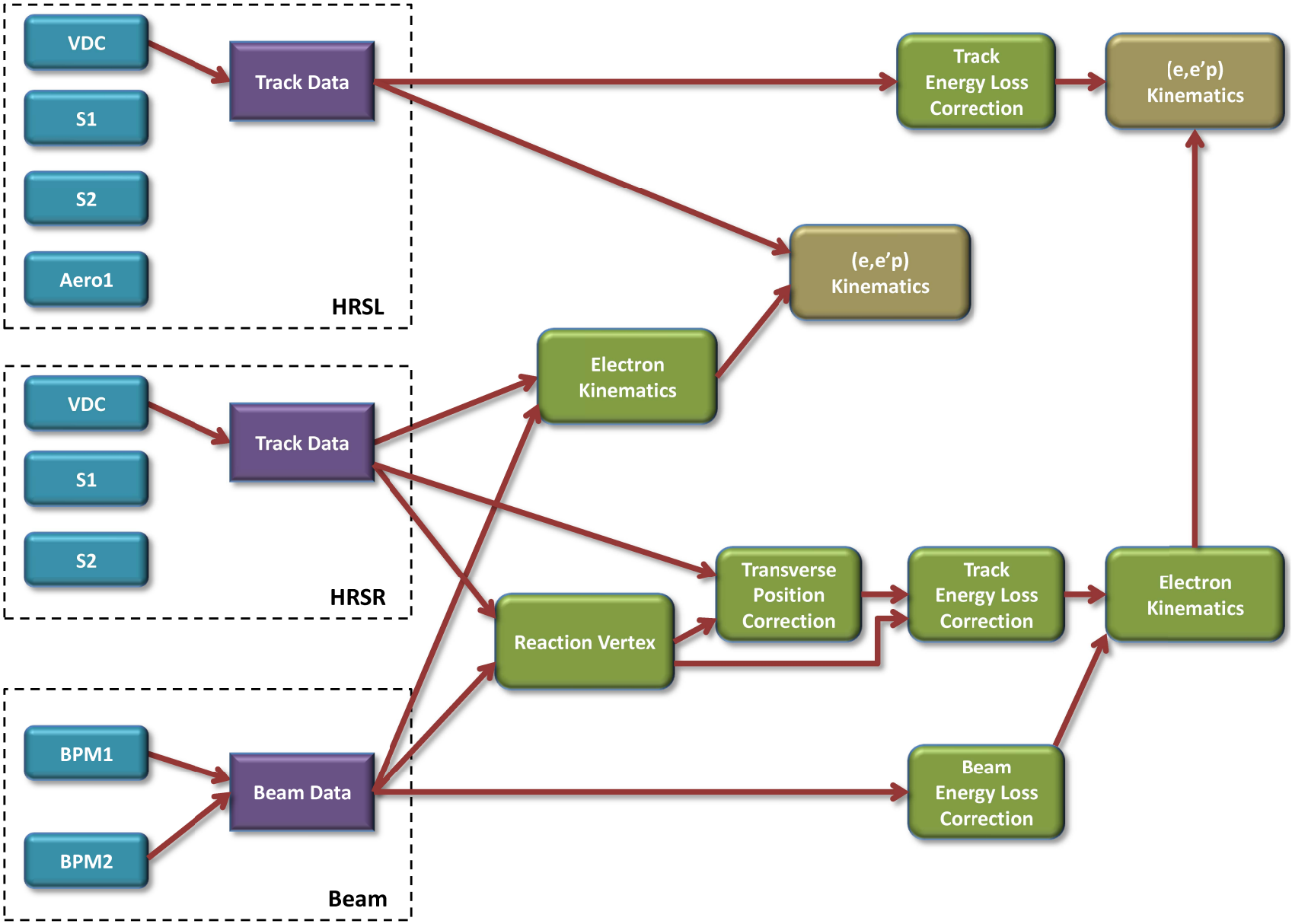
Analyzer Concepts: Analysis Objects

- Any class that produces "results"
- Every analysis object has unique name, *e.g.* R.s1
- Results stored in "global variables", prefixed with the respective module's name, *e.g.* R.s1.nhits

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

A (slightly complex) Module Configuration Example



Example Replay Script (Note: *Script* written in C++)

```
// 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->SetOdefFile("HRSR.odef");   // Define output
analyzer->Process(run);               // Process all invents in the input
```

→ See the files in `tutorial.tar.gz` for a working example

Making Sense of the Output

- Open ROOT output file with `analyzer`
- Start plotting stuff $\longrightarrow$ demonstration