Introduction: ROOT/C++ Analyzer

Hall A Analysis Workshop
10 December 2003

Development Team
Rob Feuerbach
Bob Michaels
Ole Hansen

Contributors
Bodo Reitz
Bryan Moffit
Robert Stringer (summer student)

The C++ Analyzer

- Developed by Hall A team from scratch, starting in Spring 2000
- Object-oriented. Built on top of ROOT as an interactive ROOT application
- Organized as a collection of analysis modules: apparatuses, detectors, physics modules. "Everything is a plug-in". Completely configurable to the needs of experiments.
- Easily extensible with user modules
- CODA event decoder
- Predefined event loop, initialization, etc. minimizes work for users, but users can customize if necessary
- Tests, cuts, and output can be defined dynamically without recompiling
- Centralized, time-dependent database (text files or SQL) provides clear data organization
- Almost all of ESPACE functionality implemented
- Available now

Physics Analysis Capabilities

Available in v1.0 (v1.1):

- HRS spectrometers with VDCs and all basic detectors (scintillators, Cherenkovs, showers)
- VDC tracking, target reconstruction
- Basic shower cluster analysis
- Single-arm kinematics
- Reaction point (vertex) calculations
- Extended target (x_{tg}) corrections
- Beamline with BPMs & raster
- Intelligent helicity analysis
- Scalers
- EPICS
- Scintillator timewalk corrections, beta calculation
- Coincidence time
- Coincidence kinematics

Important Concepts

- Detector
  - Code/data for analyzing a type of detector. Examples: Scintillator, Cherenkov, Shower, VDC, BPM
  - Generic design: One class describes any number of similar detectors, differing only by name and database entries
  - Must have Decode() function
  - SpectrometerDetectors must also have Coarse/FineProcess() (or Coarse/FineTrack() for tracking detectors)
  - Embedded in Apparatus or standalone
  - Must not assume presence of other detectors
- Apparatus/Spectrometer
  - Collection of Detectors
  - Generic, as with detectors
  - Must have Decode() and Reconstruct() functions
  - May combine decoded data from detectors

  - Spectrometer is an apparatus with support for tracks and a standard Reconstruct() function

- Physics Module
  - May combine data from several apparatuses
  - Typical applications: kinematics calculations, corrections
  - Must have Process() function

- Analysis Object
  - Common base class for Apparatus/Spectrometer, Detector, Physics Module
  - One optional database file (or group of entries) per object
  - Name of object is important — unique identifier
  - Provides support for naming objects, debugging, global variables, database access and similar
  - Defines common interface (API) for processing analysis objects
  - Useful virtual functions that actual objects may implement: DefineVariables(), ReadDatabase(), Init(), Clear()

- Event loop
  - THaAnalyzer
  - Handles initialization
  - Manages lists of objects, output, cuts, decoder, file names, etc.
  - Not required. Can create custom event loop, even in script
- Decoder
  - THaEvData
  - Decodes raw CODA event buffer into crate/slot/channel data
  - Supports Fastbus & VME
- Run objects
  - THaRun, THaOnlRun
  - Stores run info, e.g. raw data file name, time, run number, etc.
  - Stores global run parameters, e.g. beam energy, target mass, etc.
  - Provides access to CODA file or ET data

My Very First Script

```
gHaApps->Add( new THaHRS("R", "Right HRS") );
myrun = new THaRun("/data/e12345_6789.dat")
analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/e12345_6789.root")
analyzer->Process(myrun)
// now inspect results in memory, or quit
```

Requires:

- Database
- output.def

Resources for Getting Started

- C++ Analyzer documentation at:

  http://hallaweb.jlab.org/root/

- ROOT User's Guide at:

  http://root.cern.ch/

- Pre-installed C++ Analyzer on adaq machines.
  Login as adaq and type `analyzer`.
  See $ANALYZER directory.

- Example scripts in $ANALYZER/examples

- Online analysis setup for E94-107 (adaq account)

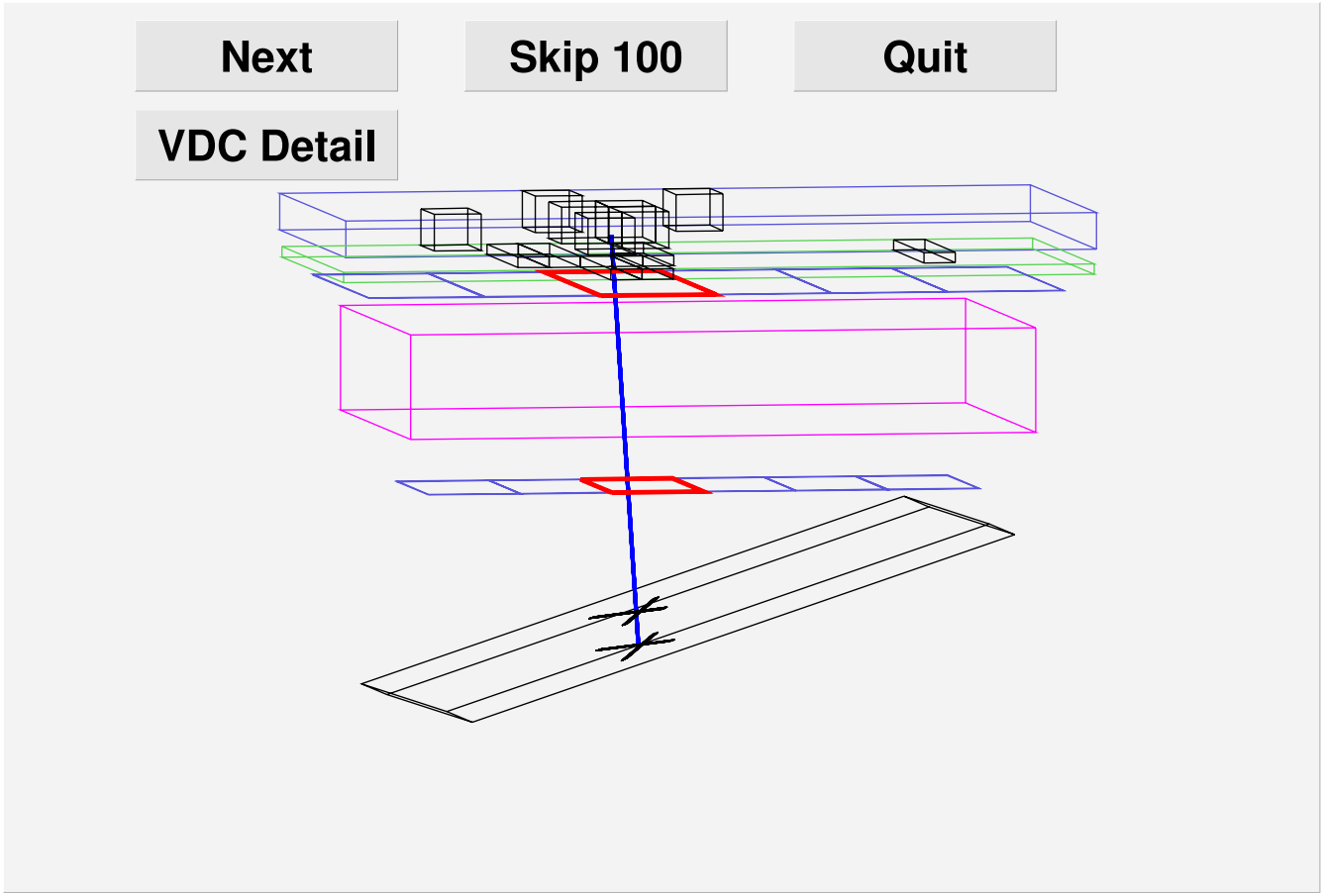
Transition Guide for ESPACE Users

ESPACE	C++ Analyzer
kumac	C++ script (full C/C++!)
detmap	part of detector's database file
database	part of detector's database file
header files	run database (db_run.dat)
reaction string	(configuration of physics modules)
masses.dat	(parameters to physics modules)
variables (block_data_var.f)	global variables (gHaVars)
logicals	cuts/tests (cuts.def, gHaCuts)
spectra	output definitions (output.def)
ntuples	ROOT tree
paw/paw++	C++ Analyzer (=interface to ROOT)
optimization	(not yet available)
hacking/recompiling ESPACE	writing user module(s) $\rightarrow$ shared lib
private ESPACE version	private shared library
(no programming docs)	auto-generated HTML ref. manual
difficult to integrate new detectors	trivial (std det), easy (new det)

Event Display

- Summer student project (Robert Stringer, CSLA)
- Prototype
- 3D visualization of detector stack
- VDC clusters
- Active elements of scintillator and shower counters
- Reconstructed track(s)
- Detailed VDC display (planes, clusters, hits)

Event Display



Physics Modules

- Perform “post-reconstruction” analysis.
  May combine information from several apparatuses
- “Plug-in” design. Use & combine as needed.
- Allow rapid extension of analyzer
- Typical applications: kinematics calculations, vertex finding, corrections, coincidence time, statistics
- Special applications: debugging, event display, custom output
- Any other event-by-event usercode.
  (But: direct analysis of actual hardware data should be done by a detector.)
- Output from modules that correct tracking info can be used as input for modules that operate on tracking info, etc.
- Modules may export variables, define histograms, etc., that can be written to output ROOT file

Current/Future Physics Modules

Kinematics	Single-arm	THaPrimaryKine
		THaElectronKine
	Coincidence	THaSecondaryKine
	Photoproduction	–to do–
Vertex	Single-arm	THaReactionPoint
	Two-arm	THaTwoarmVertex
		THaAvgVertex
Corrections	Extended target	THaExtTarCor
	Energy loss	–to do–
	Kin. broadening	–to do–
Timing	HRS coincidence	THaCoincidenceTime
Special	Debugging	THaDebugModule
	Golden Track	THaGoldenTrack
	Event Display	–prototype–

Current limitation: Only operate on Golden Track

Suggestions for additions welcome

My Very First Script

```
gHaApps->Add( new THaHRS("R", "Right HRS") );
myrun = new THaRun("/data/e12345_6789.dat")
analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/e12345_6789.root")
analyzer->Process(myrun)
// now inspect results in memory, or quit
```

Requires:

- Database
- output.def

How to Set Up a New Analysis

- Write C++ **analysis script**:
  - Set up experimental configuration (apparatuses & detectors)
  - Define desired "physics" analysis chain, if any (e.g. kinematics, corrections)
  - Define run(s) to be analyzed
  - Define input & output file names
  - Configure additional parameters, if any (e.g. file compression, number of events, reconstruction flags, etc.)
  - Optional: Start analysis
  - Optional: Post-processing
  - Optional: Loop over runs
- Create new time-dependent database directory & review/update **database files** for all used detectors
- Create **"run database"**
- Define **output**
- Optional: Define **tests & cuts**

Details on the Web

Extending the Analyzer

The core Analyzer is treated as a library that users do not normally modify. All experiment-specific extensions are incorporated through external libraries dynamically loaded at run time.

If new modules (detectors, apparatuses, physics modules) are needed:

- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility functions provided in core Analyzer (e.g. for database access)
- Create database files for the module(s)

Once experiment-specific library is written, using it is a simple matter:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH" ));
```

To help you get started with building user libraries, get the “Software Development Kit”