NOvA Software and FMWK

A simple framework for HEP data analysis and reconstruction

http://enrico1.physics.indiana.edu/fmwk

Mark Messier
Indiana University

Fermilab Neutrino Experiments Computing Workshop
March 12,13, 2009

NOvA Specifics

- NOvA will operate three detectors to search for electron neutrino appearance in a muon neutrino beam
  - An Integration Prototype Near Detector
  - A near detector underground adjacent to the MINOS detector
  - A far detector located in Ash River, MN
- Timeline:
  - IPND next year
  - First data in near and far detectors in 2012
- Number of users
  - Currently 28 institutions, 111 physicists 7 of which are graduate students.
  - Expect numbers to be comparable to MINOS as experiment reaches data taking stage
  - Like MINOS, large fraction of collaboration is off the FNAL site

Framework

- Reconstruction
  - FMWK (more on this later)
- Simulation
  - gnumi GEANT-3 based beam simulation. g4numi likely in future.
  - GENIE neutrino event generator. NEUGEN being phased out
  - CRY cosmic-ray generator
  - GDML geometry markup language
  - Geant3/Geant4 via ROOT's Virtual Monte Carlo for detector simulation
- Data analysis
  - ROOT (ntuple/histograms)
  - PostgreSQL database

What is FMWK? Why is FMWK?

- FMWK provides
  - Event data format
  - Input/Output
  - Standard methods to read and write user-created objects to events
  - Framework for modularized reconstruction and analysis jobs
  - Run-time configuration framework
  - Event display shell
  - Database interface tools
  - XML interface tools
- FMWK has its origins with the MIPP experiment which needed a C++-based analysis framework starting in about 2001. At that time the options were limited: GAUDI and MINOS. For different reasons, neither seemed to be a good fit.
- After being used by MIPP, pieces started getting used by other projects so I started the process of “de-MIPP-ifying” the code. Now being used by NOvA and LqAr groups at FNAL.

My goals in writing FMWK

- Main goal: Simple things should be simple. Complex things should be possible. Where there is a conflict between these, err on the side of simplicity.
- To me, keeping things simple means:
  - Think about users' (reconstruction code author) goals and how they would "like" to accomplish those goals.
  - Keep interfaces streamlined. Limit users' options but make intelligent choices. Provide exactly one "good" way to do something.
  - Avoid interface levels and layers of redirection.
  - Only implement what is needed by users. Avoid lots of "what if's".
  - Keep external library dependencies to a minimum.
  - Leverage existing tools and follow standards as much as possible:
    - Standard Template Library
    - Text parsing: XML (xerces-c)
    - I/O: ROOT

module.h/cxx

moduleConfig.xml

jobDefinition.xml

DAQ or MC Generator

data.root

ana

data1.root

histos.root

evd

root

IoModules
Maps event data onto files in read and write modes.

EventDataModel
Defines the event data format and read/write interfaces to event data

JobControl
Defines interfaces for building jobs out of “modules”. Provides executable “ana” which is extended at run time

Config
Defines interface between modules and configuration data

XMLInterface
Interface to xerces-c XML parser. Allows for easy extension of the XML tags handled and their mapping onto C++ objects

EventDisplayBase
An event display template. Integrates IoModules/ JobControl and Config into display program.

file

program

Package

FMWK Overview

Left: An FMWK workflow
Right: Supporting FMWK packages

Event format : `EventDataModel`

- All event data indexed by run/subrun/event numbers and a time stamp.
- Events are based on a collection of ROOT TFolders. Each folder represents a different stage of event processing:
  - `MC:` Output from Monte Carlo generator
  - `DetSim:` Output from detector simulation
  - `Raw:` Uncalibrated detector data
  - `RawAux:` Uncalibrated detector data. Useful if single detector element dominates data stream.
  - `Cal:` Calibrated detector data
  - `Reco:` Recostruction objects (tracks, showers, vertices,…)
  - `User:` Scratch pad (seldom used, not really required)
  - `Summary:` DST-style summary data for the event
- Event constructed to allow partial I/O of above pieces.

Event format : `EventDataModel`

- Reading:

```
void AFunc(const edm::EventHandle& evt)
{
  // MyObject is a user-defined object inheriting from ROOT's TObject
  std::vector<const MyObject*> objs;
  evt.Raw().Get("./myanalysis",objs);
}
```

- Writing:

```
void AFunc(edm::EventHandle& evt)
{
  // MyObject is a user-defined object inheriting from ROOT's TObject
  std::vector<MyObject> objs;
  // Do what you do to make objs...
  // ...
  // Copy them into the event
  evt.Raw().Put(objs, "./myanalysis");
}
```

% edm_dump /data/en/2a/users/messier/c120mc.root

```
Dump of run = 30015860.414 event = 1 file
= /data/en/2a/users/messier/c120mc.root
evt.Header()
==========================================
evt.DetSim()
|-* hits
|-* tpc
  |-* MCCHits[1406]
|-* tof
  |-* MCCHits[14]
|-* tofCalib
|-* dc1
  |-* MCCHits[36]
|-* dc2
  |-* MCCHits[24]
|-* dc3
  |-* MCCHits[20]
|-* rich
  |-* MCCRICHHits[1981]
  |-* MCCHits[9]
|-* t0
  |-* MCCHits[18]
|-* veto
|-* scint
  |-* MCCHits[9]
|
==========================================
evt.MC()
|-* kine
  |-* MCCParticles[79]
|-* vert
  |-* MCCVertexs[62]
|-* match_vtxdafit
  |-* MCMatchess[9]
|-* match_vtxconfit
  |-* MCMatchess[9]
==========================================
evt.Raw()
|-* trig
  |-* TPCEventInfos[1]
|-* bckov
|-* dc1
  |-* DCDigits[29]
|-* dc2
  |-* DCDigits[23]
|-* dc3
  |-* DCDigits[18]
|-* hcal
  |-* ADCDigits[8]
|-* mwpc1
  |-* MWPCDigits[9]
|-* mwpc2
  |-* MWPCDigits[8]
|-* rich
  |-* RICHDigits[48]
|-* t0
  |-* TOFDigits[13]
|-* ckov
  |-* CKOVDigits[96]
|
==========================================
```

Building jobs : “`JobControl`”

- Basic unit is a Module which is run as a Node inside a job

- User provides implementation of a class following the interface defined by the jobc::Module base class. eg. “MyTrackFitter.h/.cxx”

- User provides configuration data for the module in an XML file. eg. “MyTrackFitter.xml”

- With these provided they can be linked into the standard “ana” job:

  % ana -x myjob.xml -g myhistos.root -o outputfile.root inputfile.root

Sample job definition

```
<jobdoc>

  <xmlfile>
    TrkRBase.xml TPCReco.xml SPFit.xml VtxDAFit.xml
  </xmlfile>

  <link>
    Minuit  TPCResCor NumericalMethods
    Swimmer SPFit     VertexReco
  </link>

  <job name="Tracking">
   <node sequence="TrkRBase" filter="off"/>
   <node sequence="TPCReco"  filter="off"/>
   <node module="SPTrkBuilder" config="default" reco="1" ana="0" filter="off"/>
   <node module="VtxDAFit"     config="default" reco="1" ana="0" filter="off"/>
  </job>

</jobdoc>
```

Loading configuration data

Linking libraries

Building job

A sequence is a list of related modules

Run-time configuration : “Config”

- Stored in XML files:

```
<configdoc>
  <config name=”Sample” version=”default”>
    <param name=”AFloat”> <float> 10.0 </float> Sample float
    <param name=”AnInt”> <int> 2 </int> Sample integer
    <param name=”AVector”> <int> 1 2 3 </int> Sample vector of integers
    <param name=”AString”> <string> Zowie! </string> Sample string
  </config>
</configdoc>
```

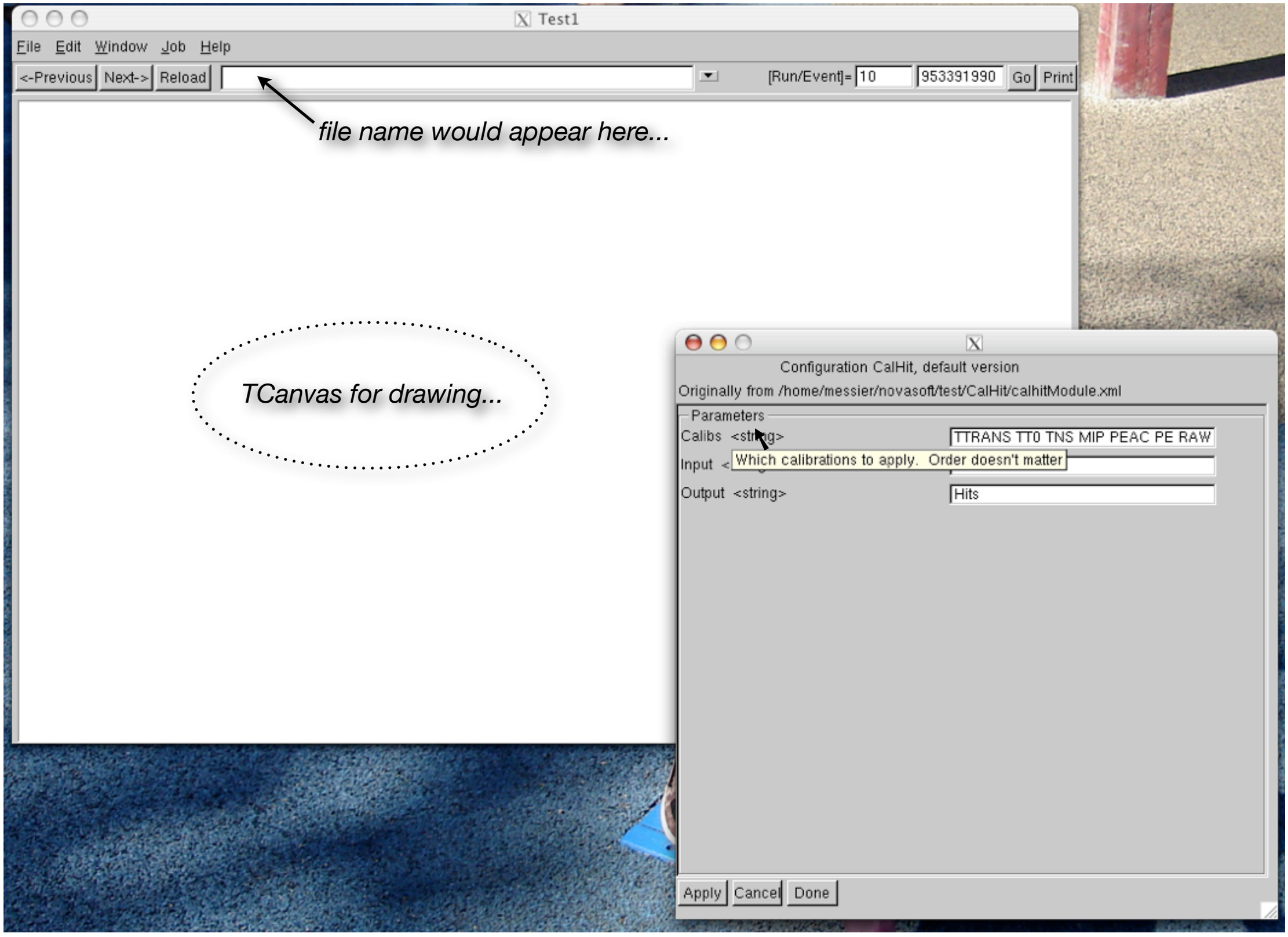
- C++ classes register themselves as clients and implement “Update” method:

```
void MyClass::Update(const cfg::Config& c) {
  c(“AFloat”).Get(fFloatMemberData);
  c(“AnInt”).Get(fIntMemberData);
  c(“AVector”).Get(fVectorMemberData);
  c(“AString”).Get(fStringMemberData);
}
```

- Framework ensures that Update() is called when configuration is loaded or modified.
- Standard GUI interface provided by EventDisplayBase

Event display tools : `EventDisplayBase`

- FMWK makes no attempt to provide an interactive shell for running jobs. All interactive use is expected to be through an event display.
- The EventDisplayBase package provides a shell which users can use to build event displays.
  - Provides a window base class with buttons and menus connecting the program to the I/O, JobControl, and Config packages as well as print utilities.
  - Provides a blank canvas on the window. Users fill this canvas in with representations of their detector and its associated data.
  - Supports multiple views of the detector data within same program
- Allows users to interactively bring up configuration data on module-by-module basis, edit, re-reconstruct, and view results.
- Of all the FMWK packages, this one is the least well developed and in my opinion the likeliest to change as it gets used more



Screen shot of the window provided by the EventDisplayBase package (program: test_evdb). The dialog window was launched by selecting “Job/Edit Config” and shows how to edit module configurations. Hitting “Apply” would cause any reconstruction to be re-applied and the event re-displayed. User’s are responsible for filling the big blank window with some representation of the data from their detector.

Full analysis chain from raw data to paper

- N/A

Full simulation chain

Goal	Program	Input	Output	Helpers
Neutrino flux calculation	gnumi	FFREAD card of geometry	Neutrino PAW ntuples	GEANT3/PAW
Neutrino 4-vector generation	gGENIE	GDML detector geometry	FMWK file of neutrino 4 vectors	GENIE
Cosmic ray rate calculation	gcosmic	GDML detector geometry	FMWK file of cosmic-ray 4 vectors	CRY cosmic ray generator
Overlay neutrinos and cosmic rays	none yet...			
Detector tracking	ana	4-vectors and GDML geometry	energy depositions	GEANT3/GEANT4/ROOT VirtualMC
Light simulation	ana	energy depositions	photons at APD	
Electronics simulation	ana	photons at APD	APD digitizations	
Initial calibration	ana	APD digitizations	T/Q Hits	
Time clustering	ana	T/Q Hits	time slice	
Spatial clustering	ana	time slice	spatial cluster	
Fitting	ana	clusters	“prongs”, reconstructed 4-vectors, vertices	
Particle ID	root	DST	mu-like / e-like / NC-like assignments	
Physics plots	root + others	DST+ others	plots	Oscillation code

Calibration procedures

- At this point, we only undo what the detector simulation does
- Charge
  - ADC to photo-electron conversion is a constant
  - Attenuation correction based on information from adjacent planes
  - Others??
- Timing
  - TDC = MC time * constant
  - t = TDC/constant

Alignment procedures

- Shown in principle using toy muon Monte Carlo, little in practice

Fortran or C++?

- C++

What works really well?

- GDML - We've gotten quite a bit of mileage out of this "write once use many times" approach to the geometry definition.
- Simulation chain seems relatively accessible. With a short list of instructions new users seem able to produce their own MC samples
- People generally like the XML interfaces for configuration and jobs and the way its been integrated into the event display
- People generally like the module interface (albeit, mostly in contrast to MINOS's equivalent concepts)
- Handling of histogram output:
  - FMWK I/O module also handles histogram output
  - Histograms automatically booked into their own subdirectory within output file based on module names (or module/version if module is not unique)
  - Accidental overwriting of histogram output prevented
  - ctrl-c or HUP signal shuts program down nicely, closing histogram files

What would we not do again?

- Several false starts with framework (GMINOS/SoCal/FMWK)
- It would be nice if FMWK allowed easier access to the event store from a root prompt for quick plots of objects stored in the events.