Fermi Offline Software: The Pros and Cons of Beg, Borrow, and Steal

Heather Kelly (SLAC)

Proposal Phase 1998

- A handful of widely distributed developers
- Gismo C++ Monte Carlo
- Hard-wired coarse geometry description
- ASCII ntuple output
- Fully integrated Event Display
- Visual Source Safe code repository
- Windows-centric development

Early Adopters

Some of our primary external library choices (Geant4, Gaudi, ROOT) were just achieving notice when we picked them up. Those choices, while risky at the time, have proven to be fruitful.

Geant 4 http://geant4.cern.ch

The Good: Great online documentation
Large user community and well vetted code
Multi-platform, binaries available
The recent move to CMake is applauded.

Drawback: During our initial migration, it took time to win over our whole team that this new Geant was as accurate as Geant3 or Gismo. We are often slow to upgrade this external.

Gaudi http://proj-gaudi.web.cern.ch/proj-gaudi

The Good: C++ Framework which includes a number of services out of the box:
Transient Data Store (TDS) – "shared" memory for data
Persistency Service
Messaging and logging
JobOptions service – runtime parameter handling
The primary flaw with our 1998 code, was poor handling of data sharing. Migrating to Gaudi achieved data and algorithm separation.

Drawback: Gaudi utilizes a large number of external libraries; we have slightly customized the source to avoid so many additional dependencies.

ROOT http://root.cern.ch

The Good: Five Star Support, Online Documentation and User Community
Modular – despite the growth of ROOT's code base, we are able to pick and choose. We primarily utilize ROOT's I/O and some of the math libraries such as TMinuit.

Drawback: complaints concerning the learning curve to produce presentation quality plots, but PyROOT provide a less daunting interface.

Not all Rosy…

Meanwhile, our choice of **CMT (Code Management Tool)** allowed us to quickly support both Windows and Linux.

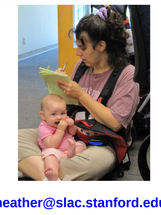
Drawback: Windows support we hoped for did not materialize. The original developer moved on to other things and there was a period with very little activity. Our solution was to freeze on an early version and ultimately move to another build tool: **SCons**. This endeavor has taken us a long time to achieve, mostly due to our need for full Windows Visual Studio support.

Why Windows?

(and cygwin just won't do)

We have a handful of proficient Windows developers attached to the Visual Studio development environment:
Integrated Debugger – go from error messages to setting breakpoints in a couple of clicks.
Integrated Editor and Build Properties – allows programmers to set compile and link settings quickly and easily.

While we rely heavily on our remaining Windows developers, it does come with a cost. Free build tools like CMT and SCons, support Visual Studio, however, they do not provide fully functional solution files which our developers demand for debugging. This necessitated painstaking work to provide. Unfortunately each version of Visual Studio requires further customization. CMake may have been an alternative, though we shied away from its custom scripting language.



SLAC NATIONAL ACCELERATOR LABORATORY

heather@slac.stanford.edu

Introduction

The Fermi Large Area Telescope (LAT) was launched as part of the Fermi Gamma-ray Space Telescope on June 11th 2008. The LAT collaboration's offline software includes:

GlastRelease: C++ Monte Carlo simulation and data reconstruction software utilized as part of the offline data processing pipeline
ScienceTools: all software related to scientific analysis of Fermi LAT data written in C++ with python interfaces

We are a relatively small collaboration with a maximum of 25 developers in our heyday. Our intent from the moment Fermi's proposal was accepted, was to provide **one** code system for simulation, test data analysis, and flight operations. During our software development, we leveraged a number of external libraries which include **ROOT, Gaudi, Geant4, CFITSIO, Swig, Python**, and **Xerces**. This helped to alleviate the lack of manpower we had available to get a system running quickly. We support a community intent on running our software on their personal laptops. This drove us to provide binary distributions and simple GUIs to aid source code compilation.
With six years ahead of us, we are in the phase where we must move forward to support modern operating systems and compilers to get us through the life of the mission. This means upgrading our external libraries as well. We have a decreased labor force, and it is crucial to our production system that we carefully orchestrate all upgrades to insure stability.

Issues And Lessons Learned

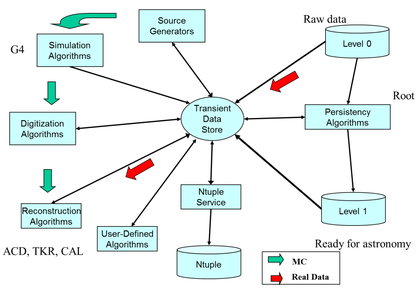
• **Using External libraries avoids re-inventing the wheel** CLHEP, ROOT, Geant4, Gaudi were all examples of code we were much better off taking and using. All supported both Windows and Linux (some also now support Mac). We could then focus our efforts on tasks specific to Fermi.
•**Support for some OSes are better than others** Windows can be a bit of a problem. Many of the externals we use now support CMake (CLHEP, G4, ROOT), making this less of an issue.
•**Use Externals sparingly** – While external libraries can offer a treasure trove of features and free code, it does come with a cost. This is code you do not control. Your ability to later upgrade operating systems or compilers may be impacted by the externals you choose today.
•**Pay attention to dependencies** – Some externals depend on other libraries. You may find that there are conflicting versions required by various externals. At best, upgrading one library, may force you to upgrade a number of others due to these dependencies.
•**Don't wait too long to upgrade** – When possible, it is much better and easier to handle incremental upgrades rather than jumping several versions at once.
•**Make Friends** – When you do utilize an external library, find the experts associated with a particular external and get to know them. You will have questions and problems associated with that external someday, and you need good resources to contact.
•**Never make use of non-standard features** – interfaces change, and certainly over the long haul of a mission, if you are taking advantage of some quirk in the code of an external library, the rug will be pulled out from under you.
•**Be wary of your own free code**– Our choice to adopt an event display built using Fox and Ruby has proved to be a maintenance issue due to the loss of both developers associated with that project. We have moved to Wired, which fortunately also uses the same HepRep protocol.

Communication Tools

At one time, we were a group of 25 developers spread across 9 time zones. In 1998, our team was spending money on teleconferences, alternate forms of communication were necessary:
JIRA - bug tracking
Confluence – Wiki tool
EVO - video conferencing
Instant Messaging, **e-mail**, and **mailing lists**

Nirvana

- Geant4 Monte Carlo
- Gaudi Framework
- XML detector definition for runtime geometry
- Full MC, detector, and reconstruction ROOT I/O
- ROOT ntuple contains summary data
- CVS repository
- Multi-platform
  - Redhat Linux, Windows, (and Mac ScienceTools only)
- **SCons** build tool http://scons.org
  - Migrating from **CMT (Code Management Tool)**
- One code system for simulation, test data analysis, and flight operations.
- Event Display utilizing the HepRep protocol
- Automated Release Manager (RM)
  - Provides binaries on all supported platforms
  - Triggered by CVS tags
- Doxygen documentation generated via RM
- Online User Workbook
  http://www.slac.stanford.edu/exp/glast/wb/prod



Stability versus Development

Our data processing pipeline has been utilizing a relatively stable version of GlastRelease since launch. Some external upgrades, patches and bug fixes have been allowed. We use **CVS** as our code repository and branching to implement required code changes to our stable releases.

Problem: There is little confidence in the use of CVS branches across our development team.
Fix: We have one or two developers willing to tackle the job of maintaining branches for GlastRelease. For ScienceTools, branches are avoided altogether in favor of rapidly applying patches along the main trunk and rolling out new tagged releases.
If our project was on a later timeline, we likely would have moved to Subversion (SVN).

Problem: Stability is often favored over introducing "unnecessary" patches, which can result in improvements being passed over for years at a time.
Fix: The development branch should keep up with external library versions more diligently, as jumping multiple versions when we do upgrade is very painful.

User Support

Our Online User Workbook, largely written by a dedicated technical writer, has been a vital component in supporting our distributed team of users and developers across the LAT collaboration. Unfortunately, we no longer fund a technical writer, but the development team is working to keep the content up-to-date.