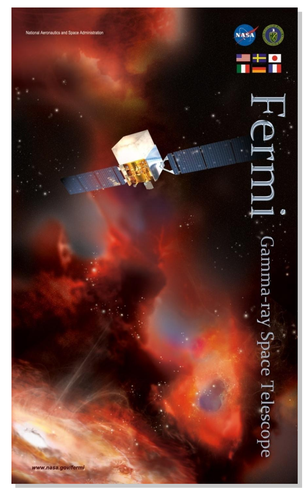
Fermi Large Area Telescope Offline Software Maintenance Madness

Heather Kelly (SLAC)



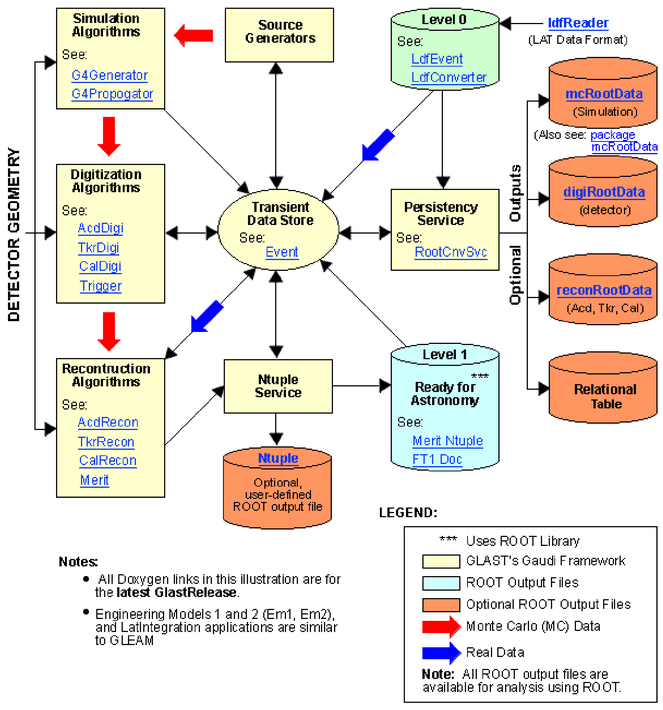
Introduction

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

During our software development, we leveraged a number of external libraries which include ROOT, Gaudi, Geant4, CFITSIO, Swig, Python, and Xerces. At launch we supported Redhat Enterprise Linux 3 32bit and Windows VS2003. With eight years ahead of us, we are in the phase of our project 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. It is crucial to our production system that we carefully orchestrate all upgrades to insure stability. This poster will focus on our experiences with two of our nineteen external libraries, maintenance of a large scale offline software project, and support of our development and user communities.

GlastRelease data flow using the Gaudi Framework



ROOT

A Data Analysis Framework

http://root.cern.ch

- Machine independent, self-describing file format
- Object Oriented I/O well suited to our OO design
- C++ interpreter for command line analysis
- Python interface (PyROOT)

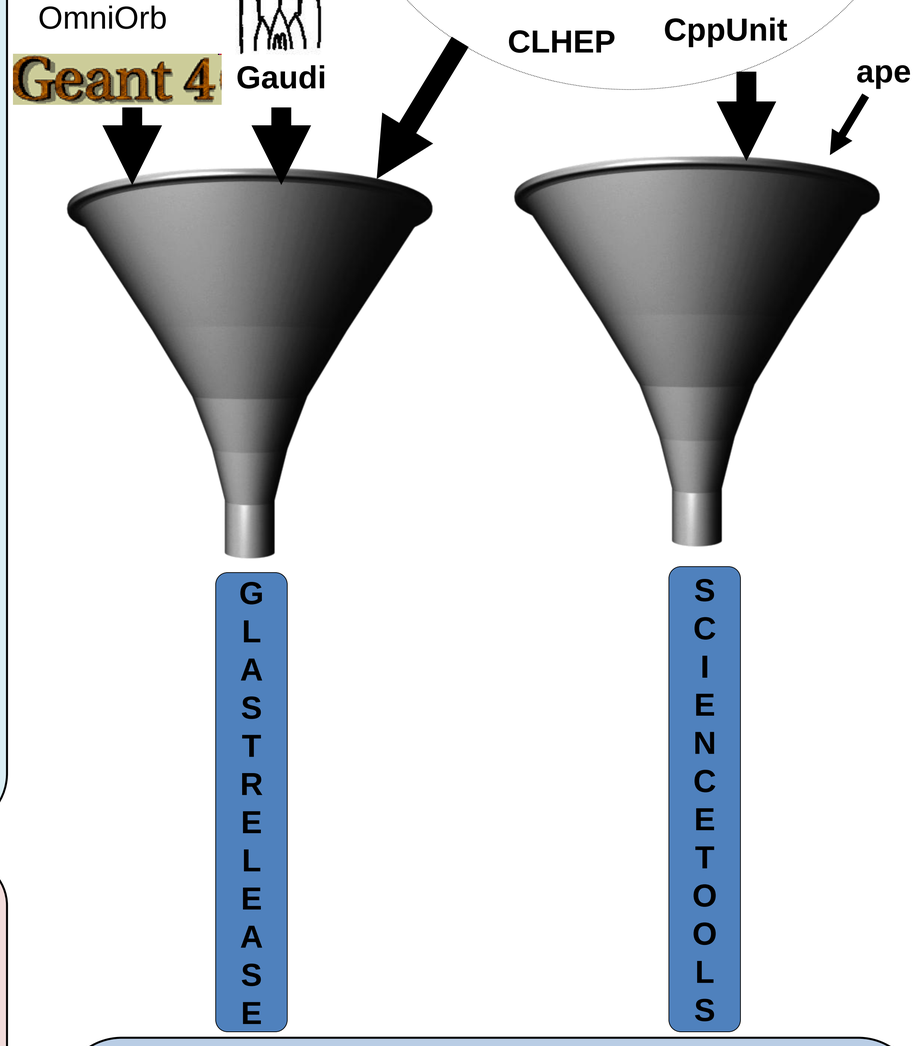
The ROOT code base has grown substantially since we adopted it as our LAT Offline file format. Fortunately, ROOT remains modular and we are able to pick and choose what portions of the framework we desire to use. In GlastRelease and the ScienceTools, we primarily utilize ROOT's I/O and some of the math libraries such as TMinuit.

Five Star Support: ROOT documentation and support are very impressive. An online user guide, active mailing lists and access to the source code provide ample aid to users. The developers make a point of addressing questions and concerns quickly. Root's support rivals commercial software, and in many cases exceeds it.

We tend to upgrade ROOT versions about once a year, where we wait a month or so after an initial production release to allow bug reports to be addressed. Today we are using v5.26.00a, the current production release.

ROOT officially supports a wide variety of operating systems and compilers, and provides binary distributions of its code. Unfortunately, we have rarely been able to take advantage of the binary distributions, due to our adoption of specific versions of python and typically requiring patch releases. Fortunately, building ROOT is fairly straight forward, though it took some learning to do it correctly on the Mac.

Complaint: Learning curve to create presentation quality plots is high.
Solution: For some, using PyROOT is less daunting.



GAUDI

Object-Oriented C++ Framework

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

"The implementation of an architecture which defines a structure flexible enough to support all types of physics data processing needs..from simulation to analysis to visualization." Gaudi Architecture Design Document

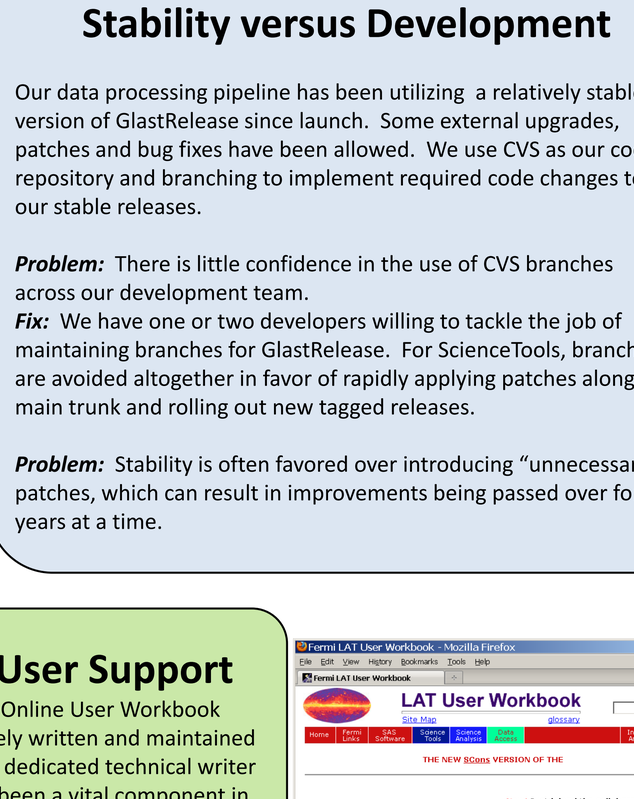
The framework provides a number of basic services:

- Event Data Service (TDS) – a fancy OO common block!
  - Messaging and Logging Services
  - Infrastructure for I/O to support multiple output formats
- Provides standard event loop
- Job parameters are handled via an input ASCII file
- Optional Python interface

We have been frozen on Gaudi v18r1 since 2006 in anticipation of launch. We are now in the process of migrating to Gaudi v21r7. This has been a time consuming process.

Problem: Provide GlastRelease support for RHEL5 and VS2008, while Gaudi officially supports RHEL4 gcc 3.4.3 and VS2003.
Problem : Conflicting dependencies between our external libraries. For example, Geant4 8.0.1 depends on CLHEP 1.9.2.2 while the new Gaudi v21r7 has moved ahead to CLHEP 1.9.4.4 (which is incompatible with 1.9.2.2).
Problem: Gaudi by default uses a large number of externals, including Boost, CLHEP, ROOT, POOL, Seal, some of which do not offer binary distributions for all the operating system and compiler combinations we wish to support. We would like to avoid having to compile these additional libraries by hand.

Fix: Use a subset of Gaudi which eliminates most of the 29 dependencies, including CLHEP. This does require some minor source code modification, but the payoff is worth it.

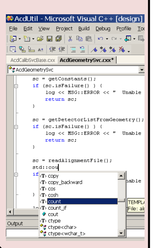


Complaint: Gaudi documentation is woefully out of date (2001), one must rely on the code itself in conjunction with the release notes to understand improvements and changes to the code in subsequent releases. There is a mailing list available, where other Gaudi users can discuss difficulties.

Supported Operating Systems

Redhat Enterprise Linux 4 – gcc 3.4.3 32 and 64 bit systems
Redhat Enterprise Linux 5 – gcc 4 32 and 64 bit systems
Windows XP (Server 2003) – Visual Studio 2003 compilers
Windows 7 (Server 2008) – Visual Studio 2008 compilers
ScienceTools Only
Mac OS X versions 10.4 (Tiger) and 10.6 (Snow Leopard)

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.
IntelliSense editing - automated class member completion.
Integrated Editor and Build Properties – allows programmers to set compile and link settings quickly and easily.

Issues And Lessons Learned

- **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.
- **Beware of your own free code**– Our choice to adopt an event display built upon Fox and Ruby has proved to be a maintenance issue due to the loss of both developers associated with that project. We are now in a situation where we are moving to another event display which does not yet provide all the features of our old one, while the old one lacks any support. Those with Fox or Ruby experience are few and far between.

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.

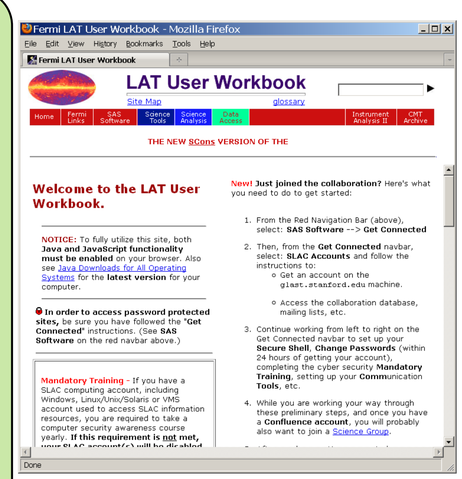
Problem: There is little confidence in the use of CVS branches across our development team.
Fix: We have one or two developers willing to tackle the job of maintaining branches for GlastRelease. For ScienceTools, branches are avoided altogether in favor of rapidly applying patches along the main trunk and rolling out new tagged releases.

Problem: Stability is often favored over introducing "unnecessary" patches, which can result in improvements being passed over for years at a time.

User Support

Our Online User Workbook largely written and maintained by a dedicated technical writer has been a vital component in supporting our distributed team of users and developers across the LAT collaboration.

We have weekly offline software meetings, as well as dedicated meetings for special projects.
Mailing lists and instant messaging also provide communication opportunities.



See other Fermi posters:
- P064 – Tom Stephens – Release Manager
- P099 – Joanne Bogart – SCons
- P100 – Richard Dubois – LAT Computing

heather@slac.stanford.edu