### Abstract
Several years into GLAST (now Fermi) offline software development it became evident we would need a replacement for our original build system, the Configuration Management Tool (CMT) developed at CERN, in order to support Mac users and to keep pace with newer compilers and operating system versions on our traditional platforms, Linux and Windows. The open source product SCons (Software Construction Tool) emerged as the only viable alternative and development began in earnest several months before Fermi’s successful launch in June of 2008. Over two years later the conversion is nearing completion. This poster describes the conversion to and our use of SCons, concentrating on the resulting environment for users and developers and how it was achieved. Topics discussed include:

- SCons and its interaction with Fermi code organization
- GoGui, a cross-platform gui for Fermi developers
- Issues specific to Windows developer support

### Use and Customization
Standard compiler options and other generic settings are added to an initial environment, but most packages clone baseline at least twice — once for an environment to build library, again for building applications — and customize the clones as appropriate. Our local extensions include:

- Add command-line options (e.g. specify non-default compiler)
- Add build tools (for Doxygen, dynamic ROOT libraries, ...) and Tools

As a matter of policy (and a switch from previous build system) we install everything needed at compile time or run-time.

### Environment set-up
SCons, particularly when combined with our decision to install all files needed at runtime in centralized locations, allowed us to streamline the distribution for remote users (easy to include intermediate build products and, for end-user distribution, package source not needed at run-time) and simplify set-up, especially for end users. End users of an SCons build can establish a process with environment suitable for running all applications by running a single set-up file; they do not need a local installation of SCons.

### Windows Project and Solution Files
The largest stumbling block in adapting SCons to make Windows builds has been the creation of adequate project and solution files. SCons native builders produce files which cannot be usefully modified in the Visual Studio environment. Generated files include an encoded version of dependencies and procedures, then invoke SCons to interpret it. The (substantial) advantage of this approach is its robustness: the build proceeds identically whether invoked from Visual Studio or directly from SCons. The disadvantage is that it is impossible to change anything about how the build proceeds from Visual Studio.

The fundamental differences between Windows and Linux make it impossible for a build system to do "the same thing" with both. In our use of CMT, the first step was to generate project and solution files. Our approach has been for the Release Manager to produce Windows builds in the standard SCons manner (no VS) and to use additional build targets to generate custom VS project files for our developers. However, capturing all the information of SCons builds and translating, bit by bit, for VS consumption is labor-intensive, unlikely to ever be perfect, and difficult to maintain.

### Issues and Lessons Learned
Conversion of the build system for a mature project is a big deal — no surprise there. In spite of the numerous shortcomings of our old system, we would not have seriously contemplated switching (nor should we) if not forced by the platform support. Pain points of the migration include:

- Windows developer support — We knew it would be difficult, but nevertheless underestimated the time it took. We’re still not quite done with Windows developer support on Windows. Overall, Windows issues have probably added a year to the project. A substantial fraction of that is for developer support.
- Change management strategy — installing all files needed at run- or build-time has significant advantages, however it wasn’t required as part of the conversion, it added to the work, and it takes some getting used to for developers.
- Tendency for other updates to get dropped in — New external library versions, new CVS tagging convention and install policy above are examples.
- Cost of supporting parallel systems — Most developers and end users are too busy to switch before they have to. Some minimal communication between systems is necessary but inevitably fickle; it’s not worth the time to do a better job.