Offline Software Development Plan and Guidelines

R. Dubois
1 INTRODUCTION

This Software Development Plan (SDP) describes the software management and development process for the Offline Software for GLAST.

1.1. Project Overview

The Offline Software comprises several components:

- Bulk production of Monte Carlo simulations
- Prompt processing of instrument data through to Level 1 event quantities
- Provide near real time monitoring information to the IOC
- Monitor and update instrument calibrations
- Create high level science products from Level 1 for the PI team
- Provide access to event and photon data for higher level data analysis
- Interface with the SSC (sharing data and algorithms)
- Interface with mirror PI team site(s) (sharing data and algorithms)

This project does not include flight software: software on the instrument for acquiring data and transmitting it to the ground.

1.2. Evolution of the SDP

This is intended to be a living, working document over the lifecycle of the project. It will be reviewed for accuracy prior to any formal reviews, whenever higher-level GLAST management policies are published to ensure adherence to GLAST standards, and when plan changes are approved which affect this document.

1.3. Definitions and Acronyms

API – Application Programming Interface
GLAST – Gamma ray Large Area Space Telescope
CVS – Concurrent Version System
CMT – Code Management Tool
IOC – Instrument Operations Center
PI – Principal Investigator
SDL - Software Development Librarian
SSC – Science Support Center
STP - Software Test Plan
TBD - To Be Determined

1.4. Document Precedence

In the event of conflict between this document and other GLAST documentation, the order of precedence for this particular project shall be:

1. GLAST Project Management Plan
2. GLAST Project Cost and Schedule Documentation
3. This document

1.5 Reference Material


2 PROJECT ORGANIZATION

2.1. Process Model
The basic development process model is described in Section 4 of this document. Exact procedures and management processes will be in accordance with the GLAST Detector Implementation Plan.

2.2. Organizational Structure
The software team is composed of a core group, building the infrastructure that the rest of the code is based upon, as well as the basic utilities that facilitate the PI team doing science analysis; a matrixed group of instrument sub-system developers who are members of both the software and subsystem groups; and a more loosely connected group of developers who will create high level science analysis tools.

3 MANAGEMENT PROCESS

3.1. Management Objectives
The primary goal of this project is to provide quality software, which is an integral part of the Instrument Team effort to provide a fully functioning analysis system meeting the requirements of the GLAST team for extracting science from the data. Management objectives toward meeting this goal are:
- Early guidance and planning of the project
- Risk Assessment and Analysis
- Incorporating configuration control procedures
- Establishing standard software development, documentation and testing procedures.

3.2. Risk Management
Risk will be analyzed throughout the project lifecycle in terms of technical, cost and schedule risks. Risk analysis shall be presented at each review, along with mitigation techniques.
3.3. Monitoring and Controlling Mechanisms

Monitoring and control mechanisms shall be in accordance with GLAST project management plans.

All designs proceed through a three-stage review process:
- Proposal for Concept
- Informal Code and Design Reviews
- Formal Reviews

All reviews are of complete packages.

3.3.1. Informal Reviews

The software developers must work as an integral part of a team with other members of the LAT to provide a tightly integrated, functional product. As part of this interaction, informal reviews will be conducted within the GLAST team of scientists and engineers. These reviews will involve a presentation of the code design and a sample walkthrough. All presentations will be stored on the web along with a review summary written by the reviewee.

3.3.1.1 Code and Design Walkthroughs

This is an informal method used to determine the completeness of a design. The designer conducts the review with attendees representing all affected interfaces. This would also be a forum for the verification of requirements and trade studies. A secondary goal is to check compliance with the adopted software style (TBD), which is important for long term maintenance of the code. These walkthroughs will be typically held within the software team.

3.3.2. Formal Reviews

Formal reviews will involve a designated committee who will report back to management on their findings. These will also involve review of the code design, but will also address the state of documentation, testing and planning.

3.3.3 Reports to Instrument Team Management

There will be monthly reports to the Instrument Team Management keeping them apprised of issues.

3.3.3. Software Configuration Management

Software configuration management includes the activities of configuration identification, change control, status accounting, and audits. The SDL will perform the function of assembling and validating releases of the code system.

3.3.3.1 Flow of Configuration Control

The software and documentation developed for the GLAST detector systems will move through distinct designations to help maintain configuration control. The state of
readiness of code packages is controlled by tags in the CVS repository. The general flow of software and documentation is:

1. HEAD Area: Area for work on code in progress.
2. Build tag: Once a developer is satisfied that particular code is ready for release, the code and documentation is given a build tag. Here the code is integrated and independently tested/operated as part of an overall system.
3. A Release is a designated collection of all code package versions representing a validated complete system.

Further expansion and definition of these areas is covered in the Technical Process section of this document.

3.3.3.2 Configuration Control Tools
CVS is to be used to provide configuration control.

3.3.3.3 Code Management Tool
CMT will be used as the code management tool.

3.3.3.4 Revision Permissions
Write access to code packages will be restricted to
   • The package owner
   • A group of developers as designated by the owner
   • A small number of administrators, to be exercise only for emergencies

3.3.3.5 Configuration Identification
The configuration identification for each code module will be the revision number assigned automatically by CVS. The package versioning scheme shall be a three number convention in the form VxRxPx, such as V1R2P2. The first number shall indicate a major version. Major versions are typically limited to when the code has undergone major core structural changes or a significant number of enhancements have been made. The second number is changed when a version has new features/enhancements. The final number indicates that bug fixes have been made without the addition of particular features.

A scheme (TBD) will provide tags to release versions of the packages and ensure that they are unique, deterministic and recoverable.

3.3.3.6 Enhancements and Changes (Corrective Action)
It is intended that changes will be orchestrated through a problem tracking system (TBD), which will facilitate submission, tracking and control of code changes. Table 1 lists the priority assignments for code changes submitted for action. Evolutionary changes in the code will be tracked via the integrated systems tests (see Section 4.3). Package owners will demonstrate the viability of new package versions before they will be considered for inclusion in the next release.

Table 1: SMR Priority Assignment
### Priority Description

<table>
<thead>
<tr>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The problem prevents GLAST from operating to its specified performance as a detector.</td>
</tr>
<tr>
<td>2</td>
<td>The problem adversely affects either an essential capability specified in the requirements or the operator’s accomplishment of that capability, and no work-around is known.</td>
</tr>
<tr>
<td>3</td>
<td>Same as 2 above, but a work-around is known which may be put in place as a temporary solution.</td>
</tr>
<tr>
<td>4</td>
<td>The problem causes inconvenience or annoyance but does not affect a requirement.</td>
</tr>
<tr>
<td>5</td>
<td>All others not falling into a category above development.</td>
</tr>
</tbody>
</table>

### 3.3.3.7 Configuration Management Documentation and Reporting

The primary reporting will be output of the bug tracking system.

### 4 TECHNICAL PROCESS

#### 4.1. Software Development

1. Software development will follow OO practices as much as possible. This will involve an iterative process: “code a little, design a little”.

#### 4.2. Design and Development Tools

##### 4.2.1. Software Processors

**4.2.1.1 Development Hardware**

The software development hardware will be Sun, Linux and Windows NT/2000 workstations.

**4.2.1.2 Target Hardware**

The software target hardware will be Sun, Linux and Windows NT/2000 workstations.

##### 4.2.2. Programming Languages

Compiled code will be written in C++. Interpreted code will be written in Java or Perl.

##### 4.2.3. Documentation Tools

All code will be internally documented to conform to the doxygen standards.

##### 4.2.4. Compilers

Compilers for C++ will be Visual C++ and gnu g++.

#### 4.3. Software Testing and Documentation
4.3.1 Testing
Testing will be broken down into two components:
1. internal histograms and statistics produced in regular ‘standard’ runs of released code under controlled conditions.
2. stand-alone ‘unit’ test programs, where appropriate, will accompany the rest of the package code in a test directory.

4.3.2 Documentation
Documentation will be tiered
- overall system use
- individual package use
- package code reference manual

4.3.2.1 System documentation
This will form the basis for a high level users guide indicating scope and usage of the major components of the system.

4.3.2.1 Package documentation
This will lay out the design of algorithms and high level usage and applicability of the package. This documentation will be bound to versions of the package in the repository.

4.3.2.1 Package reference manual
This will form a reference manual indicating the API’s of all components of the packages. This documentation will be bound to versions of the package in the repository and comes in the form of embedded comments in the code itself.

4.4. Software Development Guidelines
The purpose of coding standards is to facilitate the maintenance, portability, and reuse of custom C and C++ source code developed for GLAST systems. Guidelines are kept in a separate document (TBD).