Open XAL Software Management Guide

Introduction

Open XAL is a Java based software platform used at some of the world's most powerful accelerator facilities for running applications, scripts and services in support of conducting experiments, characterizing accelerators, simulating accelerators and performing data collection and analysis.

This guide specifies the software standards adopted for the Open XAL project.

Adoption of the guidelines described in this document ensures a level of quality commensurate with our mission to provide intuitive, reliable and powerful software in support of the accelerator physics community.

Unless otherwise specified, throughout this guide the term "software" shall refer to software developed within the Open XAL project. The term "production" is a qualifier applied to software or code provided to the customer for normal use in contrast to software under development or in the testing phase.

General Principles

Quality control rests upon the shoulders of dedicated staff and project management adhering to software management principles. Good communication and cooperation among project management, staff and customers is essential and for which no tool can substitute. The four pillars of software management resting upon this foundation of communication and cooperation are:

- Software Configuration Management for production code
- Adherence to coding standards
- Testing and validating code before releasing it for general use
- Integration of customer feedback

The following sections describe these pillars of software management in detail.

Software Configuration Management

Software configuration management provides version control such that software changes can be tracked, labeled, shared, and earlier software versions can be recovered if necessary. While a tool such as Git provides the means for version control,

This document has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.

the following procedures are followed to ensure quality control and rapid recovery if necessary.

- Production code must be under version control. Typically, this means that code is maintained under our Open XAL Git repository on Source Forge. Please refer to the Open XAL versioning document for details on managing source code in this repository.
- Code changes committed to the repository should be accompanied by an accurate message documenting the modifications.
- Code committed to the repository should be free of compile time errors and warnings. While a single compiler warning may seem harmless in isolation, the number may grow unchecked making it more difficult to debug software and determine which warnings are important in a sea of warnings.
- Release branches should be made as necessary to provide fallback options and add to the historical documentation.
- When a large software project involves team programming, the software should be partitioned into modules with a software lead assigned to manage revisions to each module. The software lead for a module acts as an expert for that module including an understanding of dependencies on the module and assumes responsibility for revisions.

Coding Standards

Adherence to coding standards allows for code sharing, eases long term maintenance and may reduce the occurrence of bugs. Many coding standards are general although the actual implementations may vary and should be interpreted to be consistent with the programming languages in use and the target environment. Except for scripts, Open XAL code is written in Java and therefore should conform to Java coding conventions. The scripting languages adopted for Open XAL are the Java variants of Ruby and Python, namely JRuby and Jython.

General Coding Standards

- The most accurate documentation is the code itself, and as such code should be self documenting including the use of verbose, unambiguous symbol names. For example, if a variable represents energy, its symbol could be “energy,” rather than an ambiguous symbol such as “x” which conveys nothing about its meaning.
- Code should be documented both internally for clarification and for public APIs as appropriate. The documentation should be maintained as code changes since unkept documentation may become misleading which can be worse than no documentation.
- Online user documentation should be provided for complex software.
- Standard naming conventions for symbols such as variables, constants, methods and classes should be followed according to the standards of each programming language. For example, Java’s naming convention calls for instance and static constants to be upper case with underscores for word separators.

- Line and character spacing should be used for clarity to offset code segments and tokens.
- Nested code should be indented (if consistent with the language) to visually offset code at different levels.
- Every literal to be used in more than one place should be assigned to and referenced through a symbol. This allows for the value to be changed in one place if necessary, avoids ambiguity of meaning and supports self documentation.
- Code should be modular for extensibility and ease of maintenance. For example, repeated code should be replaced by a macro, function or method as appropriate. Associated groups of properties should be encapsulated into classes.
- Developers should adopt accepted software design patterns such as adaptor, proxy, model-view-controller, etc.
- Horizontal integration should be used to provide proven implementations across projects and a familiar experience for customers.
- Code encapsulation should be used to present a minimal public API and hide implementation details to prevent fragile code.
- Exception handling should be used along with descriptive error messages and stack traces.
- Risks and benefits of external software should be considered. External software may provide benefits of wide adoption and community proven implementations. However, external software may also inject code that doesn't meet our quality control standards and may also introduce a dependence which cannot be easily replaced. If possible, adaptors and API wrapping should be used to limit dependence on an external API.

Desktop Applications

- All desktop applications should use the Open XAL application framework and adopt its common look and feel.
- Console and Persistent logging should be considered where a record of output would be useful for tracking functionality including use cases, errors and bugs.

Web Pages

- Open XAL documentation (internal documentation and Open XAL website) should conform to open standards so all functionality is available regardless of which modern, standards compliant browser is used.
    - HTML 5 http://www.whatwg.org/specs/web-apps/current-work/multipage/
    - CSS 3 http://www.w3.org/Style/CSS/current-work.html
    - JavaScript
- HTML, JavaScript and CSS source code should be human editable rather than generated from software.
- When attributes and features differ among modern web browsers, the common ones should be used where possible. When this is not possible, DOM should be used to determine the available calls and attributes rather than explicit reference to particular browsers as DOM will continue to work even if a particular browser's API changes.
- Purely cosmetic features may be implemented using browser specific properties as long as they don't adversely affect the performance or experience on other browsers. Such features should be limited to those which are candidates for an open standard.

This document has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.

- Use of open web standards is preferable to third party plugins where possible.
- Attachments and images should use standard web safe formats (pdf, png, jpeg).

Testing and Validation

Before software is released for production, it should be tested according to the designed use cases and the results should be validated. Several methods for testing and validation may be employed. The extent of testing and validation depends upon the use case scenarios and the criticality of the results.

One or more of the following methods may be used to test and validate software.

- Unit tests should be developed for critical code and tests run before releasing code for production. Unit tests and source code should be maintained so that code committed to the repository passes all tests.
- Prototyping software as a script (with or without a user interface) is a common technique to determine the most effective software strategy before full implementation in a more structured language.
- Benchmarks against other existing, independent solutions are effective to validate code.
- Code may be peer reviewed for accuracy (important for data analysis and scientific integrity) and robustness.
- A software module may initially be coded within a single application where it can be isolated and tested in a well defined environment before it is ported to the core for use by other applications.
- Debuggers and Profilers can be used to trace code paths and observe internal values and measure application time and memory performance.
- Staging the release of several small modifications is preferable to a single release since the individual modifications can be tested independently.
- If a widely used, large module needs to be significantly modified, a new module should be implemented in parallel with the old module so it can be adopted incrementally by applications as confidence grows. The old module can provide a benchmark for the new module.
- Offline testing should be used when possible before testing in the production environment. Options include the use of a development database, virtual accelerator, channel access readonly gateway and notional data sources.

Customer Integration

All software is created in support of one or more customers, and as such must be made to best meet their requirements within the limits of the resources available. Each site can develop its own plan for collecting customer feedback including bug reporting and feature requests. Common mechanisms for feedback may include a logbook, issue

This document has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.

tracking and direct user feedback (email or in person). Feedback which relates to the common source code should be relayed back to the Open XAL community for review and action.

Since much of our software is for research, developers are often end users of the code they author. This experience can be particularly effective in addressing software issues.

Document Revision History

This table describes the changes to this Software Management Guide.

Date	Author	Notes
September 8, 2011	Thomas Pelaia II	Draft proposal that describes the Software Management Guide for Open XAL.

This document has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.