Open XAL Project Organization

1. Introduction

This document proposes a general structure for the Open XAL project. The high-level repository organization needed to support software versioning is presented in the document *Open XAL Versioning*. This document is concerned with the organization of project source code and resources within the repository. We present a general framework with some additional options, under future revision we expect to have a final declaration.

Open XAL is partitioned into core components and extension components. The core components support the basic operation of XAL and are site independent. On the other hand, extension components are extend the basic features of XAL, are not required to be site independent, and, in fact, may be deliberately implemented for a specific facility. Figure 1 depicts the basic structure of the core Open XAL repository, beginning from the repository root node (but not including versioning). Figure 2 suggests a basic convention for site-specific Open XAL repositories.

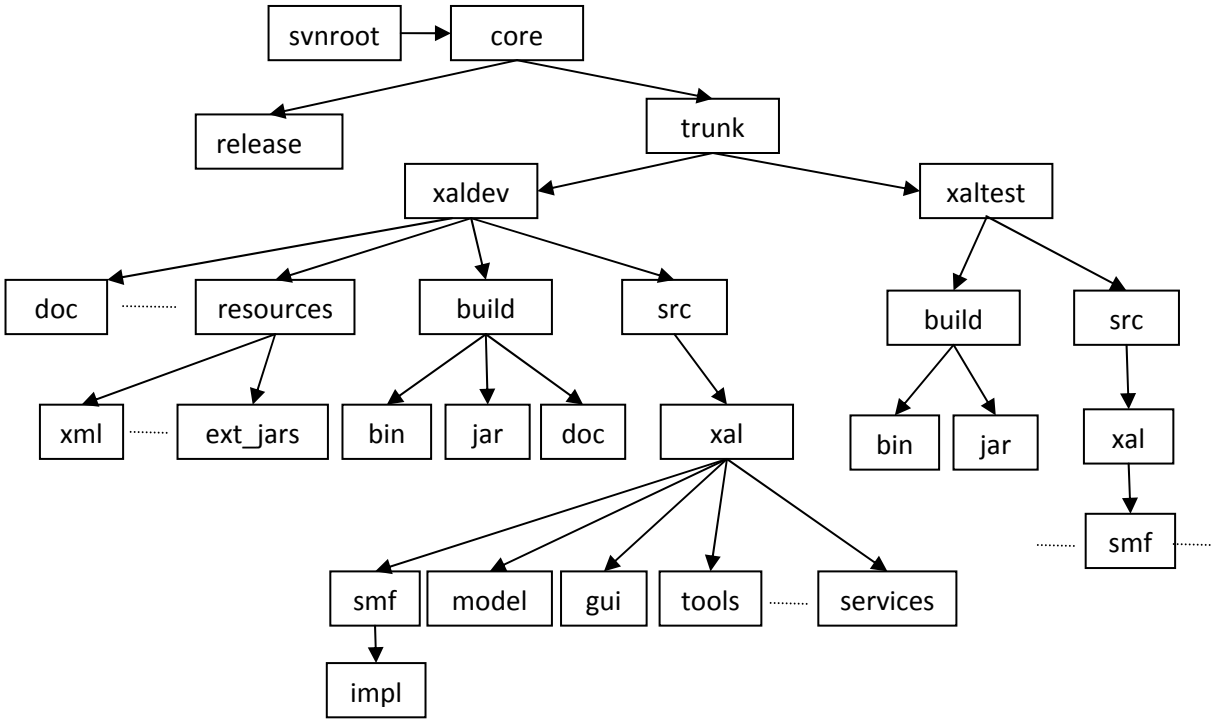

Figure 1: Proposed structure of XAL core repository

1. XAL Core

Here we outline the structure of the Open XAL core project. As mentioned, the XAL core is that portion of code and resources which is site independent. It forms the core functionality of XAL and

functions uniformly across all installations. The development location begins under the trunk node of the repository. According to the document *Open XAL Versioning*, the core development node is thus `svnroot/core/trunk`. However, we propose branching from this directory into two main directories in order to support code testing: one branch for actual project development, and one branch for project testing. This configuration is shown in Figure 1 where the trunk directory branches into `xaldev` and `xaltest`, the locations for these respective operations. This arrangement allows the test code and executables to be shipped separately from the project code and executables. Moreover, both locations are under the trunk directory so the testing code is versioned along with the actual project code which it tests.

The structure of the project directory, `xaldev`, is similar to the current XAL repository, with some notable exceptions. One is the addition of a `resources` directory where we can store project-related resources such as XML documents, configuration files, external JAR files, etc. Also shown is the high-level directory doc where we can include installation instructions, configuration information, primers, or any other documentation that we deem appropriate to ship with the core. Additionally we have also added a `src` directory, where all the source code is located, and is root of Java compilation.

Referring the to src directory of Figure 1, note that the package prefix `gov.sns.` has been removed from the current XAL repository. Since the source code here belongs to the core it should reflect the collaboration rather than an individual institution. Site specific extensions to Open XAL can prefix their source code as such. With this structure all core XAL source code has the package prefix `xal`.

The testing branch resembles the development branch. In fact, the source code locations mirror that of the development side exactly. This layout is deliberate so that the testing code can have the same Java package name as the source code which it is testing. For example, say I wish to test a new class `MyDevice` in the package `xal.smf.impl` whose source code is located in directory `trunk/xaldev/src/xal/smf/impl`. To do so I implement the test class `TestMyDevice`, this also belongs to the package `xal.smf.impl` but with source location in the test branch `trunk/xaltest/src/xal/smf/impl`. Such an arrangement makes for easy bookkeeping when designing and managing test suites.

2. Site Specific Extensions

Software that is specific to a particular facility is maintained under separate branches of the Open XAL repository as described in *Open XAL Versioning*. These branches fork directly from the root node `svnroot`. Thus, each institution has freedom to organize and develop Open XAL extensions as they see fit, so long as they do not interfere with the core. However, it may be wise to adopt some conventions to make it easier for developers to inspect and potentially use code from other institutions. With this in mind, Figure 2 portrays a proposed template for site specific project organization.

For the sake of example, in Figure 2 the repository structure for the SNS site extension is followed. Note the versioning mirrors that of the core versioning. The development trunk structure is also similar to that of the core trunk, with software development supported in the directory `dev` and software testing supported in the directory `test`. Again the src directory is the compilation root and the location of all source code. The (empty) directory `sns` lies directly below and, consequently, all Java source code has a package prefix of `sns` obviating that the source was developed specifically for use at SNS.

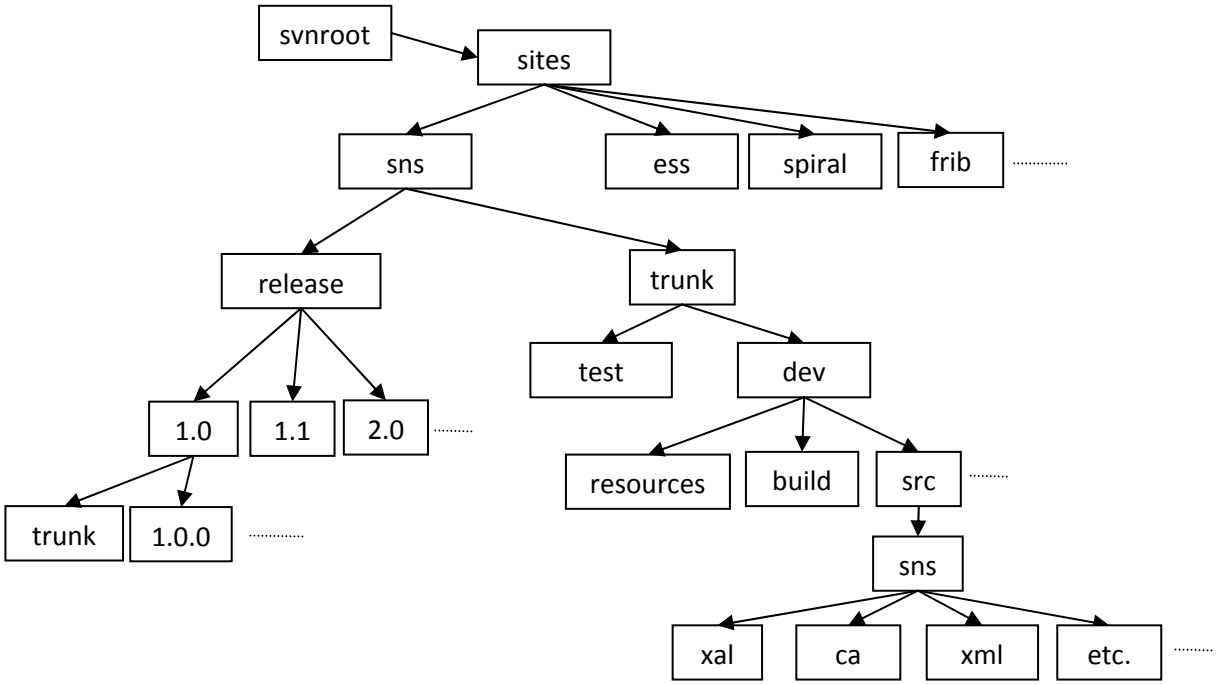

Figure 2: Suggested site specific repository structure

Other possibilities for tagging code as site would be an "XAL extensions" package prefix such as `xalext`, `xals`, or `xalx`. For example a new hardware device in use at SNS would then be supported by the class `MyDevice` in package `sns.xalx.smf.impl`.

3. Alternative Java Package Scheme

Figure 3 shows another possible scheme for organizing the Java packages and, consequently, the source branches of the XAL project. In this case we make explicit the core source and extension source code explicit by their package names. All core source code has the package prefix `xal.core` while the extension source code has the prefix `xalx` followed by the institution identifier. For example, a core hardware device supported by the class `CoreDevice` would exist in the package `xal.core.smf.impl`. A specialized hardware device particular to SNS and supported by the class `MyDevice` would live in the package `xalx.sns.smf.impl`.

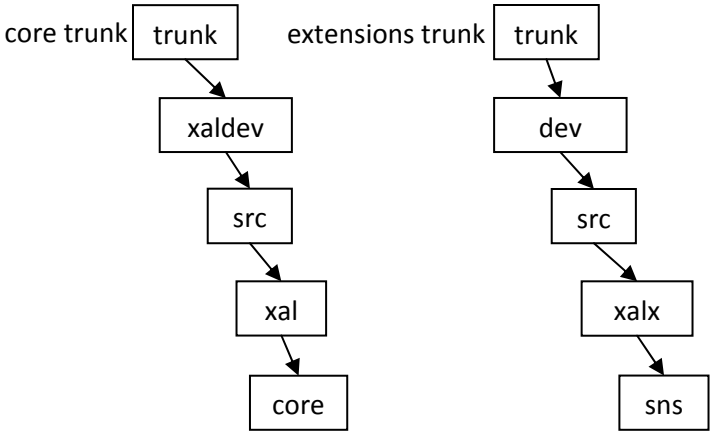

Figure 3: Alternative core/extension package structure

4. Document Revision History

Date	Author(s)	Notes
June 16	Christopher K. Allen[1]	Draft version

[1] allenck@ornl.gov