# OPEN XAL BUILD SYSTEM*

T. Pelaia II#, ORNL, Oak Ridge, TN 37831, USA

*Abstract*

Open XAL is an accelerator physics software platform developed in collaboration among several facilities around the world. The build system is implemented through Apache Ant build files and features zero configuration simplicity based on directory patterns. These directory patterns allow for correctly building the Open XAL environment including the core and site specific applications, services, extensions, plugins and resources. Options are available for deployment and custom application packaging. This paper describes the Open XAL build rules, options and workflows.

## INTRODUCTION

The Open XAL [1, 2] build system was designed from scratch to build multiple components with zero configuration while allowing for optional customization. Apache Ant was chosen as the build tool. Components include the core, extensions, plugins, services, applications and scripts. The final products are a single shared library plus the applications and services that depend on that shared library. Scripts don't require any build processing except to be copied for deployment. The goal of the build system is to provide zero configuration builds driven by convention.

## BUILD REQUIREMENTS

Apache Ant [3] was chosen as the build tool since it is mature, commonly available, well supported and currently the standard build tool for Java. The version of Ant must be at version 1.9 or later to support all of the build commands and settings used in this project. Java J2SE 7 or later is required for compiling the current source code. The Open XAL project contains all the source code and libraries required to build the project.

## ZERO CONFIGURATION

The Open XAL project delivers on the goal of building the entire project using a single command with zero configuration. Simply typing the command, "ant" in a terminal at the root of the project will build all executables. A second command, "ant install" can optionally be used for installing the executables in a deployment directory whose path can optionally be configured. Typing the command, "ant help" will display the list of all available build commands. Build files also exist throughout the project to allow builds at different levels of the tree. For example, one can build just a single application.

It is notable that zero configuration is true even when adding new components to the project. This is possible because the Open XAL build system is founded upon two principles: component separation and convention over configuration.

### *Component Separation*

Component separation means that allowable dependencies between components are restricted. This separation facilitates efficient compilation and deployment and provides well defined dependency rules. Components are categorized as the core, plugins, extensions, services, applications and scripts.

The core consists of common foundation packages and has no compile time dependencies on any other components. Plugins provide runtime support for the core. For example, a plugin could provide a branded database driver that is required at runtime for the core's generic database tools. Extensions provide additional support packages to be shared among applications, scripts and services. Plugins and extensions can depend upon each other and on the core. Applications, scripts and services [4] are the end use executables and they may depend upon the core, extensions and plugins. Applications and services may also explicitly provide extensions. For example, a service provides an extension containing the public remote interface that clients will use to communicate with the service.

While not enforced, it is encouraged that any external library be wrapped so as to abstract functionality from the details of the external library. This allows for flexibility in replacing external libraries. Furthermore, it makes it easier to identify missing components since component packages follow an Open XAL naming convention whereas external library package names are not under our control.

### *Convention over Configuration*

The build system uses convention over configuration to determine how to assemble and build components. The directory layout and naming convention determine how components are identified and how to package each component.

At the top level, component directories are appropriately named apps (for applications), core, extensions, plugins, scripts and services. A component bundle placed under any of these directories will be interpreted accordingly. Except as noted, a component

* This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC0500OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).
# pelaiata@ornl.gov

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPWI048

bundle may include any of the following subdirectories: src, resources, lib, test and extension. The src directory (not applicable to scripts) is the root of the component's Java code following the Java convention for mapping package names to directory paths. All resources, are rooted in the resources directory (if any). If any external libraries are to be included, they must be placed in the lib directory. The extension directory holds extensions if any (applies only to applications and services). Currently, only the core may also include a test subdirectory which defines unit tests. Unit tests are strictly for testing in development and are not bundled with the final products.

## BUILD PHASES

Applications, services and scripts are final executables. They depend upon a single common shared library which is built from the core, plugins and extensions. The build proceeds as follows. First, the core's code is compiled and bundled with its resources and external libraries. Next, plugins and extensions are compiled together against the core and bundled with their resources and external libraries if any. The resulting intermediate build products from the plugins, extensions and core are merged into a single shared library. Finally, services and applications are compiled and bundled with their resources and external libraries if any. By default, services, applications and scripts all reference the common shared library. An option exists to build applications and services standalone which will merge the shared library into the final build products. In this case the final executables will be larger, but each can be distributed as a single jar file.

## SITE CUSTOMIZATION

Resources associated with components can include such things as images, user interface files, online help, input data and more. Often, these resources are at least in part site specific which can pose a challenge for sharing code in a collaboration. To minimize the need to merge code across sites to customize resources, a top level site directory is supported. This site directory contains a tree mirroring the top level directory and may contain apps, core, extensions, plugins, scripts and services subdirectories. Component resources may be placed under each of these subdirectories according to the usual convention. When the resource manager loads resources, it gives preference to site specific resources of the same name if they exist.

Besides resources, the site directory can also include a build configuration file under the config subdirectory whose build properties override the defaults. For example, one can specify a different deployment directory.

## FUTURE PLANS

The current build system works very well and supports the requirements of the Open XAL collaboration. The most notable issue to address is the lack of support for unit tests outside of the core. Unit tests will need to be supported for all components. Localization of resources is another feature that would be nice to support. Integration with a dependency manager may also be considered if a compelling case can made for added value without violating the goal of zero configuration.

## REFERENCES

[1] Open XAL website: http://xaldev.sourceforge.net

[2] T. Pelaia II, "Open XAL Status Report 2015", MOPWI050, these proceedings, IPAC'15, Richmond, VA (2015).

[3] Apache Ant website: https://ant.apache.org

[4] T. Pelaia II, "Open XAL Services Architecture", MOPWI049, these proceedings, IPAC'15, Richmond, VA (2015).