

Architecture and Design for the LArSoft Continuous Integration Build System

Marc Mengel, Erica Snider, Patrick Gartung, Mark Dykstra,
Lynn Garren, Gianluca Petrillo

Fermilab

Draft version 0.3

JULy 22, 2014

Abstract

The Continuous Integration system described here will integrate the build and test of LArSoft and related experiment software with the Build Service(CIS) being setup by the Fermilab Computing Sector. This document describes the overall architecture of the LArSoft CI system, the relationship of each of the above components to each other, and a simple design to meet the requirements as described in the related requirements document.

Overall architecture

The LArSoft continuous integration (CI) system has the following major components:

- A workflow automation engine based on the Jenkins CI system¹
- A set of elementary workflows defined within the Jenkins CI configuration
- A set of “build-step” scripts called by the Jenkins workflows that drive the operations performed within each step of the workflow
- LArSoft and experiment-specific software and scripts that perform the actual tests run by the CI system.
- A hardware CI system comprised of a CI server that hosts the workflow automation engine, and a set of distributed CI slave nodes that on which the workflows are executed.

The CI server and some fraction of the CI slave nodes are provided as part of the central build / integration service operated by Fermilab SCD. ²

Deliverables

There will be several deliverables of this software system;

- A build workflow script, which will use a workflow specification and Jenkins parameters to invoke mrb and ups tools to install and/or build modules as needed to setup an test

¹The Jenkins CI system is documented in detail at <http://jenkins-ci.org/>.

²The requirements for the central build / integration service are documented in CD-DocDB document 5319. [*** Is there real documentation anywhere?? ***]

environment

- Suitable workflow specifications for LArSoft itself, and various experiment software builds
- An integration test runner tool that will be invoked by the workflow

Maintenance

The Jenkins configuration and workflow scripts are maintained in a dedicated software repository. The LArSoft and experiment-specific software and scripts live in a “test” sub-directory within the respective LArSoft or experiment repositories. The tests scripts themselves conform to a set of requirements that are described below.

Parameters

A build workflow will be triggered in a single Jenkins Project on the CIS by an HTTPS Post request, with various parameters, including:

- Revision specification. Any of:
 - None -- defaults to the “development” branch
 - global branch/release tag -- modules will be checked out with that tag if present, falling back to “development”
 - specific list of (module,tag or revision, or built-release) pairs
- Forwarded grid proxy -- this proxy will be used by the workflow to fetch input files for testing, transfer output files, etc. Various repository accounts on the repository server cdcvs.fnal.gov will need to maintain a current grid proxy file for build triggers sent from repository hooks
- Test Suite(s) -- a specific list of integration test suite names to run. Details of the integration test setup are described below.

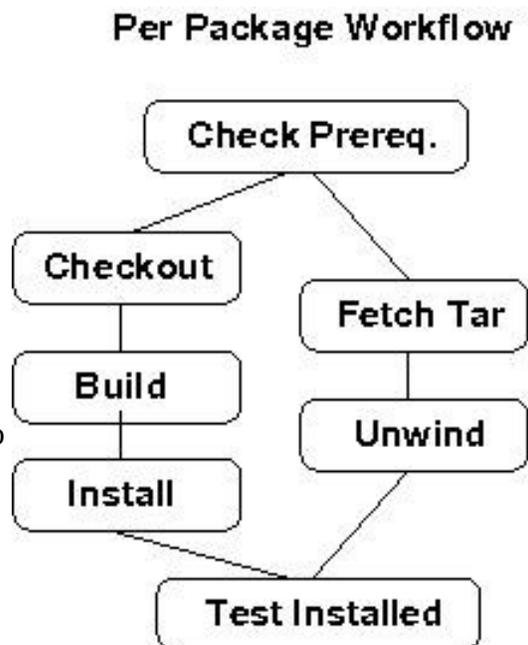
The Basic Workflow

Each sub-package follows the same basic workflow, where the version checked-out or fetched is defined by the build trigger parameters.

Workflow details

These are the things that need to happen within each step of the workflow. They will be implemented via a combination of application configuration and build-step scripts. Each section corresponds to a build-step.

*** for now, this is just a bulleted list. Probably needs more text. ***



1.1. Check pre-conditions

- Assume the work area is the current directory at build-script start-up
- Check that required resources are available
 - If not, fix or abort
- Check that externals are available
 - If not, pull externals

1.2. Check out specified code

- A wrapper build-step script must take the Jenkins trigger parameters and configure the environment so as to allow the check-out script to check out the correct code (head of a specified branch, tag, commit)

1.3. Build the code

- Wrapper needed here? (Probably not??)

1.4. Run ‘make tests’

- Wrapper script needs to translate Jenkins trigger parameters into an environment configuration that specifies information required in call to “make test” (e.g., the test group specifier)

1.5. Install the code, make a tarball of the result

- Should just need a naming convention for the tar file

1.6. Run the appropriate stand-alone tests

- Wrapper build-script configures the environment from relevant Jenkins trigger parameters (eg, the test group specifier).
- Loop through test areas
- Read test configuration file
- Discover tests to run based on test group
- Parse dependencies and test script arguments within the relevant test group
 - Noting here that allowing the test configuration file to contain arguments for the test script allows the same test script to be used within several test groups simply by changing argument values appropriately.
- Run selected tests
- Run post-test procedures