A Plan for Standard Builds of LArSoft

Draft version 5

2024-11-25

1 Introduction

For more than 30 years the Fermilab scientific software stacks have been packaged and distributed using UPS and related tools. Fermilab specific scripts and procedures were developed to build several hundred third-party software packages (i.e., software created by neither Fermilab scientific software developers nor by the experiments) into UPS-compatible forms. Because UPS is a Fermilab-specific tool, it has been difficult for non-Fermilab personnel to develop the expertise needed to contribute to building this software stack. As a result, the time taken to update versions of software, and to add new packages, is longer than it would be if the software stack were built and distributed using more widely-used tools.

In order to make it easier for non-Fermilab personnel to contribute to our shared scientific software stack, and because of the diminishing availability of personnel to maintain support of the specialized infrastructure needed, we have decided to move to using Spack as the primary tool for building and distributing the software stack. Because Spack is supported by a larger community than is UPS, we will gain the advantage of being able to use the work of others in the community who have already created the necessary recipes for building much of the software we need. We will also gain the advantage that thorough documentation is available on the web. In addition, it will be easier for members of Fermilab experiments to add new recipes for third-party software that they need to use, and to contribute those additions back to the rest of the community. Finally, Spack is better placed than UPS to handle the needs of current and upcoming experiments, as UPS is very rigid and has high maintenance overhead with respect to dependency chains, and has other limitations with regards to issues such as relocatability and release distribution.

In this document we describe a plan for how to make use of the software built, packaged, and distributed using Spack and related tools. The goals of this plan are several:

1. To allow the SciSoft team to build, package, and distribute to the experiments the software the team develops or contributes to, including the *art* framework, LArSoft, and the new framework being developed for DUNE.
2. To provide the experiments greater flexibility in building software not provided in the suite of products delivered by the SciSoft team, while also providing a clear path for sharing effort between experiments.
3. To provide more flexibility for experiments to support building their software stack on platforms on operating systems not directly supported by Fermilab, for example, at supercomputing facilities.

A central feature of the new plan is what we call a *standard build*, described in the next section.

2 Definition of a Standard Build

The move away from UPS to Spack is intended to make it easier for non-SciSoft personnel to make their own builds of software stacks. At the same time, the SciSoft team must retain the ability to build and test LArSoft, which requires building and testing the software stacks that depend upon LArSoft. Separately, the SciSoft team will also need to be able to build and test the software stack for the new DUNE framework. The framework for allowing these distinct efforts to proceed relies on the idea of *standard builds.* All builds of LArSoft (and of *art*) that will be produced by the SciSoft team will be these standard builds. The “data management” software stacks will need to package data management clients, etc., so they do not have conflicting dependencies with the scientific software stacks. It is inconvenient for users of the data management tools (e.g., *rucio-clients*, *sam-web-client*, etc.) to have to “unsetup” their experiment’s software environment in order to “setup” the data management tools. Having the tools built with the same underlying libraries will allows them to be active in the same shell session.

We distinguish two types of standard builds:

1. A standard build of a *package* (usually consisting of all the software in a single repository, e.g., for LArSoft), which is built with a specific set of variants[1]. For ex-

[1]In Spack, a variant is a way to specify build options or configuration choices for a package. Variants allow the user to customize how a package is built and installed, giving flexibility to enable

ample, if one experiment requires ROOT with support for Apache Arrow, and another requires support for Graphviz, then the standard build of ROOT would include both the Apache Arrow and the graphviz options in the standard build. It will be up to the collaborating LArSoft experiments to determine the set of options to be included in the standard build of each package.

2. A standard build of a *suite*, which is comprised of consistent standard builds of all the packages in the suite. The SciSoft team will create a Spack environment for each standard build of a suite. All standard builds of packages that are part of a suite will be pushed to an appropriate binary build cache[2] and also installed in CVMFS. The SciSoft team will make available environment definition files for each standard build of a suite through CVMFS, or `https://scisoft.fnal.gov`, or both.

3 The Plan for Standard Builds and Releases

Here we describe the "steady state" plan for releases, the standard builds that will be created for each release, and guidance on how the experiments should use them. Some adjustments to the process might be needed during the transition from UPS to Spack.

1. New releases will be created when one of the LArSoft experiments or the SciSoft team submits a successful pull request (PR) on one or more of the LArSoft repositories, or when one of the underlying dependencies is updated, which may occur at the discretion either of the experiments or the SciSoft team. The code of the experiment making a release request must be consistent with the current LArSoft release.

   PRs from experiments to recipes for 3rd party packages, and to the *art* and LArSoft packages will be welcomed. The procedure for handling such PRs will be described in a separate document.

2. For each new release of the *art* or LArSoft suites, the SciSoft team will create a number of standard builds. Each of these builds will use a specific, single source code version for each of the packages in the software stack. First-party software will be built in both *debug* and *profile* modes. Third-party packages for which *debug* builds are useful will also be built in both *debug* and *profile* mode. Others will be built in *release* mode only (i.e., in the native build mode for the package). Each package will be built using a small number of supported compilers (and specific versions of those compilers). The LArSoft collaboration and the SciSoft team will together decide on compilers to be supported.

3. A Spack environment will be created corresponding to each standard build. Users of the standard builds will be able to use `spack env activate` (or the alias, `spacktivate`) to activate the standard environment, and also build their software against that standard environment. Users who are developing all or part of LArSoft itself will be able to set up the standard environment, then build the relevant parts of LArSoft in addition to their own experiment's software stack.

4. Experiments that are part of the LArSoft collaboration should use standard builds, preferably one of the *most recent*, as the base for their own development builds. Doing so makes it possible to test new releases of LArSoft against the experiment code. If an experiment builds does not use a standard build of LArSoft, then testing by the SciSoft team becomes infeasible, which risks wasting effort creating standard builds that do not work.

5. Experiments that are part of the LArSoft collaboration are encouraged to use *one of* the standard builds as a base for their own production builds. This need not be the most recent standard build, since the pace of releases for production is determined by other factors than the pace of the releases of the LArSoft suite.

6. Experiments may choose to build alternative builds of LArSoft. The SciSoft team will be available for consulting on such builds on a best-effort basis.

4 The use of the LArSoft and Experiment CI systems to verify LArSoft Releases

Changes to LArSoft code are made via PRs to the relevant repositories. PRs that pass a review and testing process are merged into the main body of code, where they can be tagged and used in a release. The review process uses the CI system to trigger builds and two phases of tests that are built into each of the LArSoft and experiment code repositories. (In the following, we refer to the build and tests collectively "CI tests".) As in the past, the SciSoft team will rely on the results from these CI tests to determine whether the pull request behaves in the expected way. Only when test results are understood can PRs be accepted. Details of the CI testing process, the conditions that must be met for a PR to be accepted, and

or disable certain build options and configuration choices, choose dependencies, or set specific configuration parameters.

[2]A *binary build cache*, often shortened to *build cache*, is a location in which pre-built libraries, etc., are kept in the form of tarballs. A `spack install` command command can download and untar the already-built package, rather than downloading source code and building the package.

the responsibilities on various parties in maintaining the tests are described below.

1. The SciSoft team will use the LArSoft CI system to build and test the development head of each LArSoft repository as needed, such as when triggered by a PR to a LArSoft repository from any source (the experiments or SciSoft).
   Each PR or collection of associated PRs to LArSoft repositories must pass all CI tests for all LArSoft repositories. In addition, PRs must meet a set of additional requirements before they can be accepted. Those additional requirements will be proposed, documented and maintained separately by the SciSoft team. It is the responsibility of the submitter of the PR to fix test failures address issues related to the other requirements, or to arrange to have them fixed.

2. Once the CI tests for LArSoft are passed, the CI tests for each experiment that uses LArSoft are run. As in the case of the LArSoft CI tests, all CI tests for all experiments must succeed for a PR to be accepted. It is the responsibility of whatever experiment submitted or requested the PR to fix CI test failures or make arrangements to have them fixed.

3. Note that commits to an experiment's repositories, and not only changes to LArSoft, might break that experiment's CI tests. It is the responsibility of each experiment to maintain their CI system by keeping the branches of the repositories they use for development up-to-date with the development head of LArSoft. If an experiment does not keep their CI tests up-to-date with the most recent release of LArSoft, or that branch otherwise becomes incompatible with the most recent release, then their CI tests will be removed from the workflow of the CI system used to verify new PRs in LArSoft. It will be re-enabled in the workflow as soon as the experiment updates their code to work with the most recent release of LArSoft.

4. Bug fix PRs for old releases of LArSoft can be accepted for declared production releases of LArSoft only. A PR that fixes a bug in a production release of LArSoft will be merged into the bug-fix branch for that release only if the CI tests for the production release of LArSoft and the experiment passes, or the PR breaks no CI tests that passed immediately before the PR. Each experiment is expected to keep a branch in their own repositories for CI testing of bug fixes for each of the LArSoft production releases they use.

5 How we will organize Spack environments

5.1 Layered spack environments

The SciSoft team will create *layered spack environments* for our own use and for the use of the experiments. This is the technique we use to control what spack will attempt to build for each environment. By installing a consistent set of packages into a given layer, we ensure the use of those packages in all "higher level" environments, while still allowing experiments to replace any portions of the the dependency graph of packages when they have a special need to do so. Figure 1 shows for illustrative purposes how spack environments might be layered. A description of the layers shown in this figure is provided below. Details of the layers and their contents may change in the final implementation. Note that the packages that are named are for illustration only; they are not exhaustive.

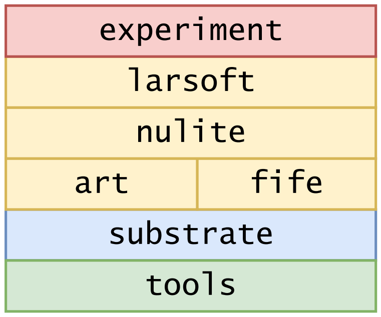

Figure 1: An illustrative example of the proposed spack environment layers. Note that *art*, *fife* and *nulite* are also used by experiments and projects that do not depend upon LArSoft. The final implementation may differ.

1. The *tools* layer includes tools that are used for development. Examples include compilers, *git*, *cmake*, and *ninja*.

2. The *substrate* layer includes products for which we (CSAID) are not in control of the source code. Examples include *ROOT*, *Geant4*, and *Catch2*. *Python* should probably be included here because some code needs to link against the *libpython.so*.

3. The *art* layer includes all of the products that are currently part of the *critic* suite. These are all products that are developed by the SciSoft team.

4. The *fife* layer includes products, excluding *art*, needed to build higher-level layers, along with data management and authentication client packages. These products are developed by other groups in CSAID. One example is *ifdhc*.

5. The *nulite* layer includes products that are needed to build *larsoft* but which are not part of *art* or *fife*, and which are controlled by other groups in CSAID. Examples are *ifdh_art*, *GENIE*, and *nusimdata*.
6. The *larsoft* layer includes all of the LArSoft products.
7. The *experiment* layer indicates the software of one LArSoft-using experiment, and any 3rd party products that are used only by that one experiment. Each experiment layer is independent of other experiment layers.

5.2 Building and distributing layered environments

The SciSoft team will be responsible for building and distributing releases of the *tools*, *substrate*, and *art* layers, while the Data Management team will be responsible for building and distributing the *fife* layer. Releases of *tools* and *substrate* will be made as needed in response to requests for newer version of packages in those layers. Updates to the *tools* layer will be made in response to requests from experiments or the SciSoft team, not automatically when new versions of the source code for those packages are released. Updates to the *substrate* layer will be made in response to requests from experiments or the SciSoft team, or when needed to support new versions of higher-level layers. The requesting party (experiment or the SciSoft team) is responsible for demonstrating that the update is self-consistent in that the updated *substrate* builds without error. The requesting party is also responsible for demonstrating that the higher-level layers also build and pass CI tests without error.

Updates to the *art* layer will be made as needed, either because of new versions of *tools* or *substrate*, or due to pull requests for bug fixes or new features in the *art* suite. The SciSoft team will remain responsible for ensuring that the *art* layer builds and passes CI tests without error.

The SciSoft team will build and distribute releases of the *fife* layer, if needed, or reuse recent builds from Data Management, who will be responsible for maintenance of the code for the packages in that layer. Similarly, the SciSoft team will build and distribute releases of the *nulite* layer, but maintenance of the code for the packages in that layer remains the responsibility of the authors of that code.

The SciSoft team will build and distribute releases of the *larsoft* layer as described above.

Each experiment retains the responsibility for building and distributing its own *experiment* layer.

It is not yet determined whether a given Spack installation would host multiple layers and/or releases of same, and what the complications/advantages of such might be. It is also not yet determined how many binary build caches we will need and when different caches should be used.

6 Other Notes on the Use of Spack

Because spack has not yet reached the 1.0 release, backwards compatibility is not guaranteed when moving to a new version of spack. The 1.0 release of spack is expected within the calendar year. At least until that release, we expect that upgrades to spack will require re-building a full new software stack.

In order to reduce unnecessary rebuilding of the software stack, experiments are encouraged to use either the appropriate Spack installation from CVMFS, or to use SciSoft-provided scripts and/or procedures to produce an appropriately versioned, patched (where necessary), and configured Spack installation.