THE HEP SOFTWARE FOUNDATION (HSF)

HSF-TN-2015-XXX
November 3, 2015

HSF Packaging Working Group Report

L. Sexton-Kennedy[1], B. Hegner[2]

[1]*FNAL,*[2]*CERN*

Abstract

The note describes the outcome of the discussions in the HSF Packaging Working Group. It summarizes the discussion on existing configuration and build tools and the possibility to converge on more common solutions.

1 Introduction

Software development in high energy physics follows the paradigm of open-source software (OSS). Experiments as well as the theory community heavily rely on software being developed outside of the field. The number of such third party software (so-called *externals*) used within a given context can easily reach over 100 interdependent software packages.[†] Creating a consistent and working stack out of 100s of packages, on a variety of platforms is a non-trivial task. Within the field multiple technical solutions exist to configure and build those stacks, in the following these will be referred to as *build tools*.

Furthermore, quite often software has to be ported to new platforms and operating systems and subsequently patches to the individual externals need to be created. This is a manual and time consuming task, requiring a very special kind of expert knowledge.

None of this work is experiment specific and our working group agrees that this effort is being duplicated. The aim of the HSF packaging working group is to see whether a better synergy within the field on porting/patching and build tools is possible.

This document describes the discussions and findings of the working group.

1.1 The Working Group

The working group consisted of members of the HEP community from the energy and intensity frontiers as well as accelerator modeling. The group met in a series of meetings over the months of May and June [1]. Additional discussions took place via the public packaging working group mailing list.

2 The HEP build and packaging tool landscape

Software stacks are needed by many organizations most notably HEP experiments. If a package is authored by the organization it is not considered an external package however it still will need build recipes. Any system that builds external packages into a coherent whole must also be capable of building the internal packages as well. In the area of external packages it will be useful to distinguish between “foreign” and “domain” external packages. Domain packages are authored by groups within the field of HEP and therefor the HSF could influence the behavior of the domain packages. The foreign packages have to be dealt with as they exist. Examples of foreign packages include: gcc, fftw3, eigen and lapack. Examples of domain packages include clhep and ROOT. In the following a few of the solutions used within the community are summarized. For more details we refer to the project websites.

[†] A package in this context is a revision controlled collection of source files including files to provide an implementation of how to build the source into binary products. These can either be object libraries or executables.

2.1 Solution 1 – Worch

Worch [2] is the build tool being proposed for the DUNE collaboration. It has been used to build the software suite common to many liquid argon experiments, called larsoft, as well as the experiment specific code that builds on top of larsoft.

2.2 Solution 2 – LCGCMake

LCGCMake [3] is the build tool used for creating the LCG releases used by the LHC experiments ATLAS and LHCb. It is based on a set of CMake macros and each external package is described using one of these macros in a semi-declarative style.

2.3 Solution 3 – cmsBuild

cmsBuild [4] is the build tool used for making releases of CMSSW for CMS. It orchestrates the building of the whole stack from the compiler on up on an as needed basis for any particular release/architecture combination. It is based on rpm build, python scripts, and text files that specify dependencies, library names and other optional information.

2.4 Solution 4 – contractor

Contractor is the build tool used by the accelerator modeling community and was developed for the Synergia project. The project must support a wide diversity of platforms from laptops to super-computers. Looking at the requirements of this community and the solutions they chose to meet them should inform the HSF community as we look forward to a time when many more diverse platforms will be need in HEP computing. Contractor has no external dependencies and is implemented in Python.

2.5 Solution 5 – SciSoft (mrb/ups/cetbuildtools)

MRB is an acronym for Multi-Repository-Build and is used by some of the intensity frontier experiments at FNAL. Unlike other tools in this list it requires ups (an acronym for Unix-Product-Support) to create an environment for it to operate in properly. cetbuildtools is a set of scripts that drive CMake to build an interdependent set of packages. The resulting packages are uploaded to a distribution server called SciSoft.

3 Taxonomy of non-HEP tools

3.1 Solution 6 - Homebrew

Homebrew [5] is a MacOS specific build tool. Packages build instructions are declared by writing Ruby classes with a given set of attributes. The feature set of these declarations is similar to those of cmsBuild and lcgcmake. A project fork focuses on the addition of Linux as a supported platform.

3.2 Solution 7 - Nix

Nix [6] is a cross-platform build tool. Package build instructions are written in the *Nix expression language*.

4 Tool Features and Requirements

We compared the feature set of the tools described. In evaluating them we defined a set of criteria in which to measure the tools against each other. However, one experiment's strict requirement may just be considered a feature for another experiment. In the following we explain the comparison criteria and the situation for the individual tools.

4.1 Supported Platforms and Environments

The most basic criterion is the support for the various platforms and environments used in high-energy physics. There are three categories of environments:

1. **Linux**
2. **MacOS X**
3. **Windows**

The first category is split into several flavours of distributions. The main distribution are the RedHat derived Scientific Linux [7] and CentOS [8]. Compared to these, Debian [9] and Debian based based distributions play a smaller role in computing centres. On desktops, the Debian based Ubuntu [10] seems rather popular.

In addition, multiple hardware architectures are in use or will be in use in the foreseeable future - x86, ARM, PowerPC, MIC, and various dedicated super-computers. Some of them impose the requirement of cross compilation onto the build and packaging solution, which is why the keyword **Xcompiler** is added to the platform table. The table 1 summarizes this information for the tools considered by the working group so far. Both Linux and MacOS X are supported by all of the tools, while Windows is not supported at all. X-compilation does not seem a priority of the projects yet.

4.2 Build and Installation Variants

One important feature of build tools in HEP is the support for the installation of multiple stacks or package versions in parallel.

1. **Multi-Rel**: The support for building and installing multiple stacks in parallel. This is important if multiple versions of experiment software need to coexist on a given installation.

	Linux	MacOS X	Windows	Xcompiler
cmsBuild	+	+	-	+
Contractor	+	+	-	?
Homebrew	+	+	-	-
LCGCMake	+	+	o	o
Nix	+	+	o	o
SciSoft	+	+	-	?
Worch	+	+	-	o

Table 1: Supported platforms of the different build tools. A yellow "o" denotes that the support is untested or not of production quality.

	Multi-Rel	Multi-BuildVar	MultiShell-RTE	Relocation
cmsBuild	+	+	🟩	+
Contractor	+	+	?	?
Homebrew	-	-	NA	-
LCGCMake	+	+	+	+
Nix	+	-	+	-
SciSoft	+	+	+	+
Worch	+	+	+	+

Table 2: Support for multiple releases and build variants. *NA* denotes that the criteria are not-applicable for homebrew. As it only provides one variant it can rely on default system paths.

2. **Multi-BuildVar**: The support for multiple versions of the same package within the same stack. There are many possible reasons why this maybe needed, what they have in common is the need to specify compiler switches that have to be applied consistently across the build, like a debugging option or a compiler dialect.

3. **MutiShell-RTE**: The support for setting up the runtime environment for a given stack version, supporting multiple shell flavours.

4. **Relocation**: Whether the build tool supports the relocation of packages or creates fully relocatable packages.

The assessment of the tools according to these criteria are listed in Table 2.

4.3 Ease of Install and Use

The criteria in this category are:

1. **Depends-On**: the dependencies of the build tool itself.

2. **Ease-Add-Pkg**: the ease of adding another package to a stack.

	Depends-On	Ease-Add-Pkg	Ease-Bootstrap	Documentation
cmsBuild	Python,rpm,apt	spec-file	o	-
Contractor	Python	?	?	-
Homebrew	Ruby	Formula	+	+
LCGCMake	Python, Cmake	Cmake-macro	+	o
Nix	Perl	expression	-	+
SciSoft	Cmake	Cmake-macro	o	-
Worch	Waf,Python	Text-files	+	o

Table 3: Ease of use of the various build tools.

3. **Ease-Bootstrap**: the ease of bootstrapping the build system itself. A prerequisite is the possibility to use it w/o root privileges.

4. **Documentation**: existence and quality of documentation

The findings are summarized in Table 3. The dependencies of most of the tools seem well under control. The addition of new packages is of similar complexity in all cases. Only the ease of bootstrapping differs. For most of them it is a simple checkout. However, others like Nix an entire (fake-)root environment is necessary. The documentation of the domain specific packages is in general very poor, the documentation of the non-HEP tools surprisingly good and complete.

4.4 Other Criteria

During the meetings proponents brought up other creiteria that are not so easy to group together. Some might consider these requirements, other might disagree. For now we've just listed them as "other". The criteria in this category are:

1. **Performance**: The relevant metrics are a. length of time to build the entire stack, this includes support for parallel builds. b. length of time to incrementally build a developer defined subset of the stack.

2. **Sys-Reuse**: the ability of the build system to reuse parts of the system software that is being built if desired.

3. **Community**: whether the tool is used by a wider community or just by one collaboration

4. **Unique-IDs**: A method of uniquely identifying a build product such that if it already exists, it does not have to be built again.

5. **VCS-Support** Integrated support for check-out of given tags or branches from the projects repositories directly.

The findings are summarized in Table 4.

	Performance	Sys-Reuse	Community	Unique-IDs	VCS-Support
cmsBuild	+	-	-	+	+
Contractor	?	?	?	-	?
Homebrew	+	+	+	-	+
LCGCMake	+	+	+	+	+
Nix	+	-	?	+	?
SciSoft	o	?	+	?	+
Worch	+	+	-	o	+

Table 4: Further considered features of the build tools. Details in the text.

5 Sharing of porting/patching – "Build recipes"

At the core of all packaging systems sits the execution of a given set of build instructions for a particular package, so-called *build recipes*. Framed by collaboration-specific pre- and post-processing steps. The behaviour of these build instructions may be platform dependent or they may be influenced by parameters like build type or paths to their dependencies.

A lot, probably even most of the work in the packaging infrastructures goes into porting software and preparing these *recipes* for the various platform-dependency-compiler-version combinations. When comparing the (implicit) interfaces of the packaging systems towards their individual recipes, they are strikingly similar. Thus starting with sharing these recipes looks like a viable solution. Various options have been discussed on the HSF packaging github tracker [11]. A first implementation of this idea was done in the context of the ALICE experiment [12].

6 Summary

Together with most of the software librarians of the HEP community we assessed the existing build and packaging tools. For this we identified 17 criteria, which are however of different importance to different users. The goal was to see whether there is a base for working on a future common build tool.

While in general of high quality, the non-HEP tools we looked at seem very weak on the support for multi-stack, multi-configuration setups. There are however more tools in the open-source community we did not investigate on yet. Among the HEP specific tools examined so far, *LCGCMake* currently seems the most suited candidate for generalization. If decided, it could serve as the tool for the *HSF Reference Builds* of the projects participating in the HSF. A more ambitious goal would be finding a common tool for building experiment stacks.

In parallel to the assessment of existing build tools, we discussed the concept of shared *build recipes*. As one o the next steps in the working group we will try to lay this out in more concrete.

Acknowledgements

We would like to thank all people participating in the constructive and lively discussions. In particular Jim Amundson, Marco Clemencic, Giulio Eulisse, Ben Morgan, Pere Mato, Shahzad Muzaffar, and Brett Viren.

References

[1] Indico agendas of 25.2.2015, 2.6.2015, 9.6.2015, 16.6.2015, 23.6.2015

[2] Worch: `https://github.com/brettviren/worch`

[3] LCGCMake: `http://ph-dep-sft.web.cern.ch/document/using-lcgcmake`

[4] cmsBuild: `https://github.com/cmsbuild/cmsdist`

[5] Homebrew: `http://brew.sh/`

[6] Nix: `https://nixos.org/nix/`

[7] Scientific Linux: `https://www.scientificlinux.org/`

[8] CentOS: `https://www.centos.org/`

[9] Debian: `https://www.debian.org/`

[10] Ubuntu: `https://www.ubuntu.com/`

[11] Packaging Protocol Discussion: `https://github.com/HEP-SF/packaging/issues/1`

[12] The alidist package: `https://github.com/alisw/alidist`.