THE HEP SOFTWARE FOUNDATION (HSF)

HSF-TN-2018-01
June 22, 2018

HSF Platform Naming Conventions - A Proposal

B. Hegner[1]

[1] *CERN*

Abstract

The note describes a proposal for a common platform naming scheme for HEP and tools to automate the platform identification.

1 Rationale

Multiple ways of denoting hardware platform vs. compiler vs. operating-system vs. build type combinations exist throughout the field of high energy physics. Quite often packages with different naming conventions have to be combined and a significant amount of time is spent to transform between these different conventions. Furthermore, this transformation is error prone and has to be updated from time to time when new hardware platforms or compiler implementations arrive.

2 Proposal

In the view of the fact that naming conventions are just agreements we propose to adopt existing naming conventions wherever possible and useful.

We consider the following pieces as part of the convention. A build configuration is denoted by the scheme

$$architecture - OS - compiler - buildtype$$

where *architecture*, *OS*, *compiler*, and *buildtype* are described in the following. Furthermore, we propose a common tool that helps identifying and building these strings.

2.1 Architecture

Different OSes may call the same *architecture* by different names. We propose to rely on the abstraction of the Python package *platform*, namely `platform.machine()`. This translates to, e.g., x86_64 for current Intel and AMD CPUs. In cases where optimized builds for a certain processor generation are required and software is not runable on the generic architecture, the generic architecture should be replaced by *architecture+instructionset1+instructionset2....* For example *x86_64+avx2* denotes an x86_64 processor supporting the AVX2 instruction set. There exist no explicit conventions for the instruction set across operating systems, thus we propose to create ad-hoc conventions. In the Linux case the *flags* field of */proc/cpuinfo* could be used. The instruction sets mentioned should be ordered alphabetically to ensure consistency.

2.2 Operating System

The operating system (*OS*) is a combination of the name of the operating system itself and its major version, e.g. *ubuntu15* or *slc6* in case a canonical abbreviation exists.

2.2.1 Linux

The name and version are as given by Python's `platform.linux_distribution()`, using the short name of the distribution, transformed to all lower-case. This leads to names like *ubuntu, centos, fedora.*

nota bene: Due to mistakes in the distribution of Scientific Linux CERN 6, this convention identifies SLC as redhat. This is fixed in the tool mentioned further below.

2.2.2 MacOS X

The name is set to *macos* and the version as given by the first two parts of `platform.mac_ver()`, e.g. yielding *macos1010*.

2.2.3 Windows

The name is set to *win* and the version as given by the first two parts of `platform.win32_ver()`.

2.3 Compiler

The *compiler* is a combination of compiler name and compiler version. We propose to use the self-given names of the compilers like *gcc, clang, msvc, icc* and adding the compiler version indicating feature relevant versions. That can be major, minor version, and patch version numbers for LLVM (e.g. *clang350*) or just major and minor version for GCC (e.g. *gcc73*). In case the system compiler is being used, the compiler should be denoted as *native*.

2.4 Buildtype

The *buildtype* denotes whether there is a debug build, an optimized build, or any other special setting. We propose to use *opt* for 'optimized' and *dbg* for debugging builds. The buildtype can be used to specify further custom build options, e.g. the used C++ standard, leading to *opt+std14* or *dbg+std18*. In general, any special compile time setting can be tracked in this name component.

2.5 Generic Cases

In some cases the mentioned components do not influence the build-artifacts created or are not relevant. In those cases, the corresponding name component is replaced by the string *all*. Examples are pure-Python packages that do not require compilation at install time.

3 Examples

`x86_64-centos7-gcc511-opt`
`x86_64-slc6-clang350-dbg`

4 Identification tool – hsf_get_platform

As to simplify the adoption of these conventions, we provide a minimal command line tool *hsf_get_platform*, that is able to identify architecture, operating system, and compiler in a transparent way. It can be used to, e.g., auto-derive everything in a given environment or to extract one part of the platform string
`hsf_get_platform.py --buildtype opt` → `"x86_64-ubuntu15-gcc49-opt"`
`hsf_get_platform.py --get compiler` → `"gcc49"`.

5 Possible Extensions to denote Platform Compatibility

The proposed naming convention identifies a given platform. In many cases, different platforms, however, may behave the same or yield compatible binaries.

For example, *slc6* is compatible with *sl6* and *redhat6*. Similarly, patch versions of compilers are *usually* compatible with each other.

We provide an additional tool `hsf_platform_compatibility` to tell the compatibility of platforms. The logic and mapping of these compatibilities cannot be derived from first principles and thus has to be maintained actively. In the long view, the maintenance burden should be shared across the community. It can be used like follows

```
hsf_platform_compatibility.py x86_64-slc5-gcc481-opt
x86_64-slc6-gcc481-opt
```
→ `"OS'es slc5 and slc6 are incompatible"` + *return code 1*

6 Resources

Both of the tools mentioned are available at `https://github.com/HEP-SF/tools`.

Acknowledgements

We would like to thank Ben Couturier and Brett Viren for fruitful discussions and feedback on earlier drafts.