

ROOT Packaging Survey

Results and discussion

ROOT

Data Analysis Framework

<https://root.cern>



The questions

1. What can the **ROOT** development **team** do that would **help** you with packaging ROOT on your distribution?
2. Is **builtin**, patched **LLVM/Clang** an issue for you? Why?
3. How important for you is it that ROOT uses **system packages** for its dependencies in general vs bundled dependencies (ROOT builtins)?
4. Which **options** do you **enable** when **building** ROOT for your distribution? Do you rely on **default** values or do you **specify** a value for all options?
5. Do you foresee any issues in continued maintenance of ROOT in your distribution if it moves to require **C++14** or later?
6. Which **sustainability issues** do you foresee for ROOT in your distribution? That is, what would it take for ROOT to still be maintained in your distribution 5 years from now?



1. What can we do to help?

In general, the current status seems to be fine for most packagers. However, we can still improve in some areas:

- ▶ Enable **building** ROOT modules **separately** (PyROOT, TMVA, RooFit cited multiple times)
 - Better **isolation** for CI environments, improved testing
 - **Shorten** builds
 - For **PyROOT**: easier to install for different Python versions (e.g. 3.8 and 3.9)
- ▶ **Share** LLVM/Clang between vanilla cling and ROOT
 - Less **security** issues

CERN login required to file bugs

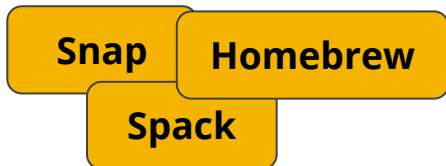


Moving to Github issues solved it

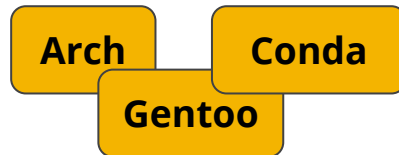


2. Is builtin LLVM an issue?

NO



YES



The general opinion is that **unbundling** LLVM/Clang would be **good**. Critical points:

- ▶ **Distribution policies** (e.g. for security)
- ▶ Too **long builds**
- ▶ External builds while still using **static libraries** doesn't improve on security



3. System Packages

Other bundled dependencies follow similar reasoning wrt builtin LLVM/Clang:

- ▶ Offer the option of **bundling** is not a problem per se
 - It shouldn't be the **only** option
 - It could lead to **clobber** files from system packages (e.g. XRootD)
- ▶ Use system packages as much as possible
- ▶ At best, **avoid** bundled dependencies:
 - **Security** flaws can be **fixed** in one single place (upstream) and **faster**
 - Lose the power of **community** for bugfixes and improvement
 - **Old** lingering **code** with no interest in updating it
 - **Waste** of **hardware** (copies of the same package occupy disk and RAM)



4. Which options are enabled?

- ▶ **Most** maintainers (with the exception being Snap) **specify** all the **options** to have more stability between releases.
- ▶ **Renaming** build **options**, or changing their default behavior can be a **burden** on package maintainers, as **bugs** can appear with no apparent changes on their part.
- ▶ Binary distributions tend to **enable** as much as possible (i.e. all options with dependencies available in the distribution are usually enabled).



5. Can we move to C++14?

- ▶ All package maintainers seem to be ok with moving to **C++17!**
- ▶ CentOS7 and older Debian versions (not present in the survey) wouldn't support the move due to old compilers
 - Only affects the official distribution package



6. Sustainability issues

- ▶ Possible issue is current maintainers **stepping down** from their role
- ▶ In most cases another person should be **ready** to step in
- ▶ Biggest effort is the **initial creation** of the package
- ▶ Maintenance is comparably easier, **if** build system doesn't change drastically



- ▶ CUDA support is burdensome (Arch separates packages, Snap doesn't support it)
- ▶ Make sure RPATH is set correctly (to avoid scripts that modify LD_LIBRARY_PATH)
- ▶ Enable roottest build against a ROOT installation
- ▶ Update CMake version (current 3.9 on Linux/MacOs, 3.16 on Windows)
 - Features: [FetchContent \(3.11\)](#), [PROCESSOR_AFFINITY \(3.12\)](#), [BUILD_RPATH_USE_ORIGIN \(3.14\)](#), [CXX_STANDARD=20 \(3.12\)](#)
 - Distros: CentOS7 (3.17), Fedora 33 (3.18), Ubuntu 20.04 (3.16), Debian 10 (3.13), Ubuntu 18.04 (3.10)



Thanks!