Building ROOT 6 for Nix

BV

January 23, 2016

Contents

1 Introduction and Overview

There is a ROOT5 Nix package but it has some problems:

- it does not actually build (fails to find X11)
- it does not not build many of the features needed (eg, databases Python, fftw3, PNG/JPEG)
- it does not use CMake despite that being the obvious intent
- it produces a monolithic output
- it's not ROOT 6

The goal here is to write a Nix package that fixes these issues while learning what are nixpkgs "best practices".

1.1 Other ROOT Packaging

1.1.1 Modular builds

ROOT itself is designed with a high degree of modularity both in terms of compile-time and run-time variants as well as providing an explicit plug-in system. This modularity can be preserved to give the end-user flexibility in which parts of ROOT to install, and more importantly, what dependencies they must be satisfied.

ROOT comes with support for building Debian and Red Hat packages, (Christian Holm Christensen). The same system builds packages for both distributions and the results are various modular packages relying on the distro's native dependency resolution system. The packaging system lives in the ROOT source under the `build/package/`.

It is desired that Nix packaging extend this existing packaging system or at the very least embrace its strategy.

1.1.2 Monolithic builds

It's typical for individuals, experiments or collectives to build ROOT from source tailored to what they need. Instead of the modular approach of the built-in packaging they simply target the necessary sub-set of features and call the whole thing "ROOT". This means one person's "ROOT" is not another person's, even if it's the same version.

This is find unless these variants are to be somehow managed together, such as in a HEP-wide packaging system. Going this route requires identifiers (Fermilab calls them "qualifiers") to be invented. Since ROOT has multiple, orthogonal build choices there is a vast number of variants to cover all combinations. Consider:

- language bindings (python, r, ruby, and maybe "go" one day)
- database support (sqlite, MySQL, postgresql, oracle)
- optional xrootd support
- optional proof support
- optional pythia support (v6 or v8)
- optional gsl support

There are more, but this is already enough to provide for hundreds of combinations. This alone is not manageable. If multiple compilers are to be supported things get even worse. And, of course, new versions of ROOT are coming out so building out this complexity is an ongoing affair.

One way to avoid these explosive combinatorics is with a targeted-monolithic strategy where a select few points in this multi-dimensional space are chosen for building. One negative consequence of this is to marginalize away any groups that require unsupported combinations. Only combinations deemed worthy by central build services are created.

2 Initial build notes

These are the steps to a monolithic ROOT 6 build in Nix, targeting just one of the combinations described above.

2.1 Starting out

It's recommended to fork `NixOS/nixpkgs` in GitHub so I work out of that as `origin`.

```
$ cd /srv/nix
$ git clone https://github.com/brettviren/nixpkgs.git
```

Now to pick a name and location in nixpkgs. The repository is organized by some category system which seems to have good intention but is actually not well suited to finding a good home for ROOT and mostly just serves to confound my attempts to find packages. In any case, the existing ROOT 5 package is at:

```
$ find nixpkgs -name root
nixpkgs/pkgs/applications/science/misc/root
```

ROOT is used outside of science (a little) and I wouldn't call Physics "misc" given that it's the basis of all science (mathematicians, you be quiet). ROOT is also not predominantly an "application" (`root.exe` is just a few dozen lines of code). Also in nixpkgs is:

```
$ find nixpkgs -name geant4
nixpkgs/pkgs/development/libraries/physics/geant4
```

That seems like a more appropriate location. Also, the name ROOT is a really horribly generic one. To distinguish it somewhat I'll pick `rootsys` and put it as a sister to `geant4`.

```
$ emacs nixpkgs/pkgs/development/libraries/physics/rootsys/default.nix
```

Also, must add to `nixpkgs/pkgs/top-level/all-packages.nix`.

```
# for now, minimal
rootsys = callPackage ../development/libraries/physics/rootsys {
};
```

The build environment always starts with this:

```
$ proot -b /srv/nix/nix-1.9-x86_64-linux/:/nix bash -l
$ source $HOME/.nix-profile/etc/profile.d/nix.sh
$ cd /src/nix
```

2.2 Getting the source

To start, let's get a recent release of ROOT6 source on disk and in an environment to build it.

```
$ nix-prefetch-url http://root.cern.ch/download/root_v6.04.02.source.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   343  100   343    0     0   1625      0 --:--:-- --:--:-- --:--:--  1633
100 94.6M  100 94.6M    0     0  5841k      0  0:00:16  0:00:16 --:--:-- 7231k
path is '/nix/store/bbfq8x7hmk521xspxc1iv0b7r26rcgsa-root_v6.04.02.source.tar.gz'
16irxlpl5xirz4v5mnnfs672j6v1j21lmf4xjrjzabjrllvmwhc1
```

Type that in to the recipe (see `ba216cd89f4d434167a24090e078dd69ce8d1ed3`) and test:

```
$ nix-build /srv/nix/nixpkgs --pure -A rootsys
```

It should download, unpack, try to patch, run configure (thanks to the existence of ROOT's fake autoconf script) and then fail.

Can also exercise these first bits by hand:

```
$ nix-shell --pure -A rootsys /srv/nix/nixpkgs
$ cd /srv/nix
$ unpackPhase
$ ls root-6.04.02/
```

2.3 Initial CMake'ing

Next is to configure the source with CMake. Nix cues off of the existence of ROOT's `configure` script so that needs removal. That is done by writing a little shell fragment and setting it to the `preConfigure` variable.

The CMake step can be done by hand like:

```
$ nix-shell --pure -A rootsys /srv/nix/nixpkgs
$ cd /srv/nix/root-6.04.02
$ cmakeConfigurePhase
```

It' s a mystery to me how the `cmakeConfigurePhase` instead of the default `configurePhase` gets run for you when a `genericBuild` is done, but it will.

At this point, this will fail as we do not tell the environment about any packages, in particular X11.

2.4 Specifying first dependency: ZLib

Let's keep the CMake step failing on X11 for a while, just to keep things failing fast. Instead, focus on supplying the ZLib dependency:

```
-- Looking for ZLib
-- Could NOT find ZLIB (missing:  ZLIB_LIBRARY ZLIB_INCLUDE_DIR)
-- Zlib not found. Switching on builtin_zlib option
```

Find out how ZLib is spelled:

```
$ nix-env -qa '.*zlib.*'
...
zlib-1.2.8
zlib-static-1.2.8
```

Add `zlib` to the arguments to the function in `default.nix` and `buildInputs`.

```
$ nix-shell /srv/nix/nixpkgs --pure -A rootsys
$ cd /srv/nix/root-6.04.02
$ rm -rf build
$ cmakeConfigurePhase
...
-- Looking for ZLib
-- Found ZLIB: /nix/store/az2scrkb88l2q09xa0g6lpbv2mh8lxjl-zlib-1.2.8/lib/libz.so (fou
```

It is necessary to re-start the `nix-shell` to pick up the addition of `zlib` as reflected in the environment and delete the prior `build/` dir so have CMake recheck.

2.5 More dependencies

Keep repeating the above to provide dependencies. For many things, ROOT provides the dubious option to build a dependency using source it provides. The strategy I take is to prefer system packages.

To iterate, adding more to the `default.nix` and testing one can use `nix-shell` as above repeating the `cmakeConfigurePhase` script or from outside the `nix-shell` do:

```
$ nix-build /srv/nix/nixpkgs --pure -A rootsys [-K]
...
note: keeping build directory '/tmp/nix-build-rootsys-6.04.02.drv-0'
```

This repeats unpacking the source, does it's business in `/tmp` and cleans up at the end, even on failure unless the `-K` is given. If the dregs of the last are kept around, subsequent runs will increment that trailing number.

2.6 Final version

This is the commit `4ede6c5da4ada577c6669ebfa851d4d1c72db071`.

2.7 ROOT Build problems

ROOT's CMake build is pretty good but a few things

- MySQL dependencies fail late in the build due to not finding `mysql.h`. The fix is to add this to CMake's config command (`cmakeFlags` variable):

```
-DMYSQL_CONFIG_EXECUTABLE="{mysql}/bin/mysql_config"
```

The patch which is part of the ROOT 5 Nix package is still needed.

```
--- a/cmake/modules/RootBuildOptions.cmake      1969-12-31 20:30:01.000000000 -0330
+++ b/cmake/modules/RootBuildOptions.cmake      2014-01-10 14:09:29.424937408 -0330
@@ -149,7 +149,7 @@

 #---General Build options-------------------------------------------------------
```

```
 # use, i.e. don't skip the full RPATH for the build tree
-set(CMAKE_SKIP_BUILD_RPATH  FALSE)
+set(CMAKE_SKIP_BUILD_RPATH  TRUE)
 # when building, don't use the install RPATH already (but later on when installing)
 set(CMAKE_BUILD_WITH_INSTALL_RPATH FALSE)
 # add the automatically determined parts of the RPATH
```