A Julia interface to the ROOT framework

Philippe Gras

IRFU, CEA, Université Paris-Saclay, France

Sep. 30, 2024

Introduction

- Need for integration of the ROOT framework and more generally legacy C++ libraries had been identified in Potential of the Julia Programming Language for High Energy Physics Computing, Jonas Eschle et al. (2023).
- WrapIt! was developed to automate generation of Julia-binding for C++ libraries.
  - Was presented at Erlangen's JuliaHEP workshop.
  - CxxWrap used for the C++-Julia interface.
- ROOT.jl has been rewritten from scratch using WrapIt!.

Reminder of WrapIt! goals

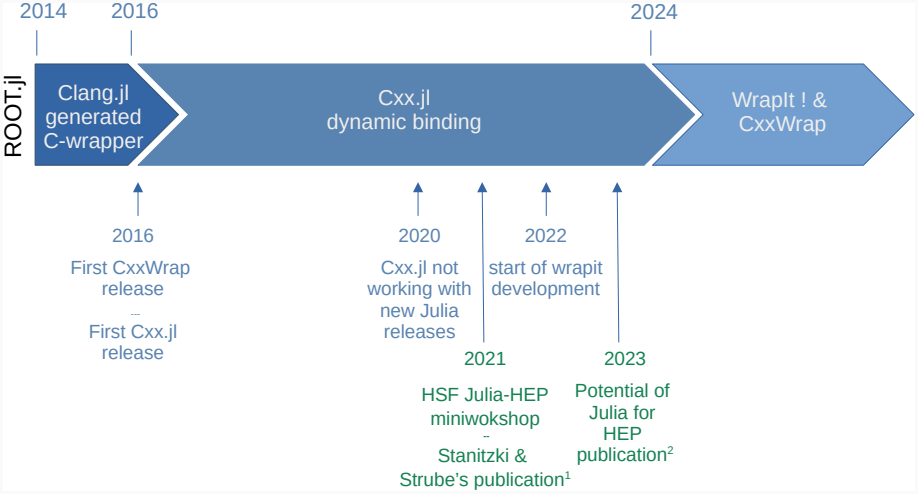
- Transparent for the Julia user:

```
say_hello("World") to call       void say_hello(const char*)
a = A()            to instantiate class A
```

```
@ccall "./libHello.so".say_hello("World"::Cstring)::Cvoid
@cxx cxx_say_hello(pointer("World"))
```

- Support for large libraries with 1000+ classes and methods.
- Minimal effort to add the bindings to an existing C++ library and update them when the library code evolves.
  - ⇒ Automatic discovery of the types and methods to bind.
  - ⇒ Requiring a compilation step is not a problem.

ROOT.jl history



Original developer: Joseep Pata; Oliver Schulz joined in 2017; Philippe Gras main developer of the CxxWrap version, with several contributions from Pere Mato. Few other developers contributed along the ROOT.jl history.

[1] doi:10.1007/s41781-021-00053-3 [2] doi:10.1007/s41781-023-00104-x

The release 0.3.0 revolution

- Second ROOT.JL revolution: first was the migration to CXX.JL and dynamic binding (ala CPPYY).
- No more limited to Julia 1.3.x ! 😀
- Static binding
  - → Limited to ROOT classes included in the build. ☹
  - → But can be relatively easily extended to more classes. 🙂
- C++ ROOT libraries are installed automatically.
  - Can also use an already existing installation, with constraints on the ROOT version.
- Package in the General Julia registry
  - → Easy installation: `julia> ]add ROOT`

Currently supported classes

Release v0.3.2

TSystem, TROOT, TInterpreter,	→ System classes
TH1x, TH2x, TProfile, TGraph, TAxis,TCanvas, TPad,	→ Histogram, graph and plotting
TF1, TF1Parameters, TFormula, TFitResults,	→ Functions and fitting
TRandom,	→ Random number generation
TFile, TDirectoryFile, TTree, TBranch,	→ ROOT I/O including TTrees
TTreeReader, TTreeReaderValue, TTreeReaderArray,	→ Reading TTrees
TObject, TClass, TNamed, TVectorD, TVectorF, TSeqCollection, TList	→ Base and collection classes

Release v0.3.3-moreclasses

```
add https://github.com/JuliaHEP/ROOT.jl#v0.3.3-moreclasses
```

All classes from v0.3.2

The full Histogram package apart from `TMutiGraph` class

The full Geometry package

Supporting more ROOT classes

Adding new classes is relatively easy

- For the easiest cases: adding the name of the class header file in the configuration file of the code generator will be enough.
- For less-easy cases, some method or types, causing issue but unneeded, will need to be added in the veto configuration file.
- For worst cases, extra development of WrapIt! needed.

Templates

- Most difficult cases are templated class, which have limited support in WrapIt! (linked to libclang limitations)

Hackatron!

Working group to prioritize the classes to add, and possibly start adding them.

Demo

Click here for the demo

Altenative link (view from web)

ROOT I/O

Providing TFile write support using native ROOT libraries had been identified as a priority at the last JuliaHEP workshop.

- Reading/Writing histograms and any ROOT class other than TTree and RNtuple is straightfoward.
- Reading TTree is easy thanks to TTreeReader.
- Writing TTree is difficult because of "SetAddress" mechanism of ROOT.

The RootIO.jl package built on top of ROOT.jl will provide a higher-level interface.
See next talk from Yash Solanki.

Handling of ROOT dependencies

Current system

- Since release 0.3.0 ROOT library and its dependencies downloaded from conda-forge if the expected ROOT release not found in the user environment.
- Compilation of the C++ part of the wrapper done at the $\mathrm{ROOT.JL}$ package installation time.

Pros of the current system

- Does not duplicate ROOT installation, if **expected** release is already installed in the user environment.
- Automatic installation of dependencies if it is not available.

Cons of the current system

- A non-standard way of managing binary dependencies of a Julia package.
- Large volume of software (4.9 GB) downloaded by Conda, which won't be shared with other Julia packages.
- Long package installation, due to Conda package installation and the compilation of the C++ part of the wrapper. $\Rightarrow$ **Limited number of ROOT classes in the default release**.

Handling of ROOT dependencies

Use of the Julia BinaryProvider, aka _JLL packages

- First discussions on a ROOT_JLL dates from Jan. 2019
- Several challenges
  - ✓ Both JULIA and ROOT use LLVM, but of different releases. Solved.
  - ✗ Need to cross-compile root on Alpine (light container-oriented Linux distribution using musl as c library instead of the more common `glicc`). Lot of progress performed but not solved yet.
  - ✗ ROOT library must be "dlopen'ed": causes recently discovered and not-yet-understood issues.
  - ✗ Handling of compiler/libc/stdc++ version dependency.
- **Will solve the package installation long time issue, by allowing to precompile the wrapper.**

Hackatron

A good challenge for the Hackatron!

Documentation: examples

Several usage example shipped with the package

https://github.com/JuliaHEP/ROOT.jl/tree/master/examples

- Histogramming, plotting, writting histogram to disk
- Fitting Histograms and Graphs
- Reading and writing TTrees

Documentation: API reference

Nowadays

Users need to consult the ROOT reference manual.

Future

Help (`docstring`) pages generated from the ROOT doxygen-based reference manual.

Hackatron!

Development of a doxygen $\rightarrow$ Julia help page convertor in Julia is an easy project.

Will be useful for several other projects.

Starting code already available.

Conclusions

- A new Julia interface to ROOT.
- Includes already many ROOT classes.
- Provides TFile writing. See next talk for a higher-level interface for TTree writing.
- Next development priority: speeding up package installation by distributing compiled c++ library as a `_jll` package.
- Several topics identified for the Hackatron: class prioritisation, cross-compiling challenge, doxygen-to-julia-docstring converter.