HSF Julia WG meeting, June 20, 2024

WrapIt! and ROOT.jl

Philippe Gras

IRFU, CEA, Université Paris-Saclay, France

June 20, 2024

Introduction

- WrapIt! is a tool that automates generation of Julia-binding for C++ libraries together with CxxWrap.
  - Was presented at Erlangen's JuliaHEP workshop.
- ROOT has been used as a testbench for WrapIt! from its early development.
- Cxx.jl-based ROOT.jl was replaced by a WrapIt!/CxxWrap-based implementation two weeks ago (version 0.3.0).

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`~~

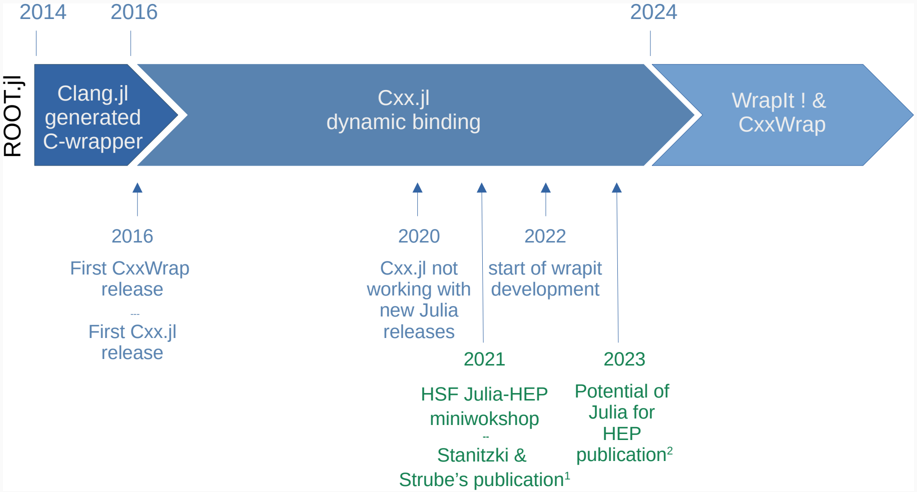
  ~~`@cxx cxx_say_hello(pointer("World"))`~~

- 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



2014
2016
2024
ROOT.jl
Clang.jl generated C-wrapper
Cxx.jl dynamic binding
WrapIt ! & CxxWrap
2016
First CxxWrap release

First Cxx.jl release
2020
Cxx.jl not working with new Julia releases
2022
start of wrapit development
2021
HSF Julia-HEP miniwokshop
--
Stanitzki & Strube's publication[1]
2023
Potential of Julia for HEP publication[2]

Original developer: Joseep Pata; Oliver Schulz joined in 2017; Philippe Gras developed CxxWrap version. Recent PRs 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 v0.3.0 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

Number of Julia bindings

82 ROOT classes/types with their methods

Main included classes

TSystem, TROOT, TInterpreter,	→ System classes
TH1x, 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

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)

Contributing

The best way to contribute to ROOT.jl is to add ROOT classes you miss.

→ Check out JuliaHEP/ROOT.jl-generator ↗, which is the code that generates ROOT.jl code and start by adding your Class`.h` in the `ROOT.wit.in` file and run `julia --project=. generate.jl`.

Examples and tests (1/2)

Examples

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

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

Tests

We have only the examples as tests (run in the github CI[1])

- Given the extent of ROOT features, difficult to add unit tests for each of them.
- Could be handled by porting ROOT tutorials. Nevertheless, tutorials are difficult to test beyond testing they don't crash.
- Could define some use cases and develop unit tests for them, but difficult to be exhaustive

[1] Continuous integration system

Examples and tests (2/2)

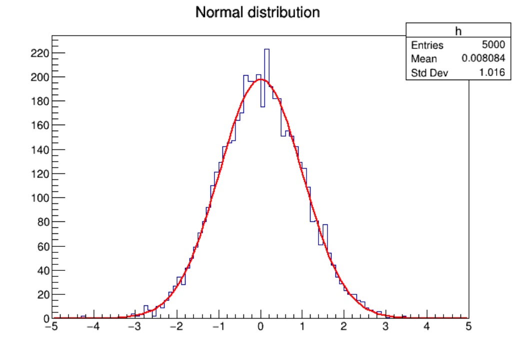
Contribution opportunity

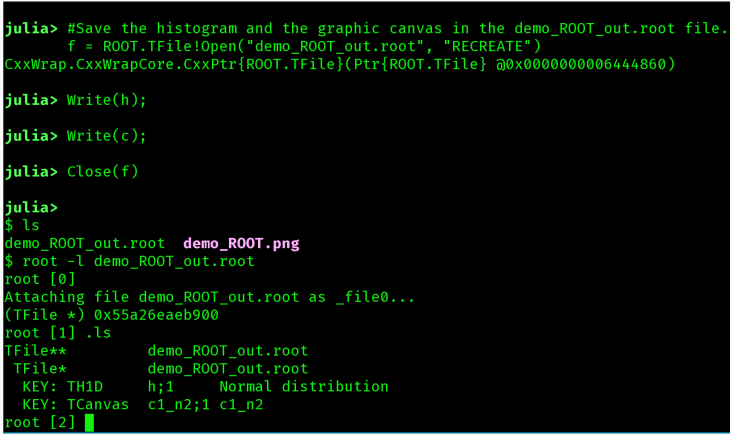
Port a ROOT tutorial ↗.

Example (1/2)

```
using ROOT

# Create a ROOT histogram, fill it with random events, and fit it.
h = ROOT.TH1D("h", "Normal distribution", 100, -5., 5.)
FillRandom(h, "gaus")

#Draw the histogram on screen
c = ROOT.TCanvas()
Draw(h)

#Fit the histogram wih a normal distribution
Fit(h, "gaus")

#Save the Canvas in an image file
SaveAs(c, "demo_ROOT.png")

#Save the histogram and the graphic canvas in the demo_ROOT_out.root file.
f = ROOT.TFile!Open("demo_ROOT_out.root", "RECREATE")
Write(h)
Write(c)
Close(f)
```

Example (2/2)





```
julia> #Save the histogram and the graphic canvas in the demo_ROOT_out.root file.
       f = ROOT.TFile!Open("demo_ROOT_out.root", "RECREATE")
CxxWrap.CxxWrapCore.CxxPtr{ROOT.TFile}(Ptr{ROOT.TFile} @0x0000000006444860)

julia> Write(h);

julia> Write(c);

julia> Close(f)

julia>
$ ls
demo_ROOT_out.root  demo_ROOT.png
$ root -l demo_ROOT_out.root
root [0]
Attaching file demo_ROOT_out.root as _file0...
(TFile *) 0x55a26eaeb900
root [1] .ls
TFile**         demo_ROOT_out.root
 TFile*         demo_ROOT_out.root
  KEY: TH1D     h;1     Normal distribution
  KEY: TCanvas  c1_n2;1 c1_n2
root [2]
```

ROOT I/O

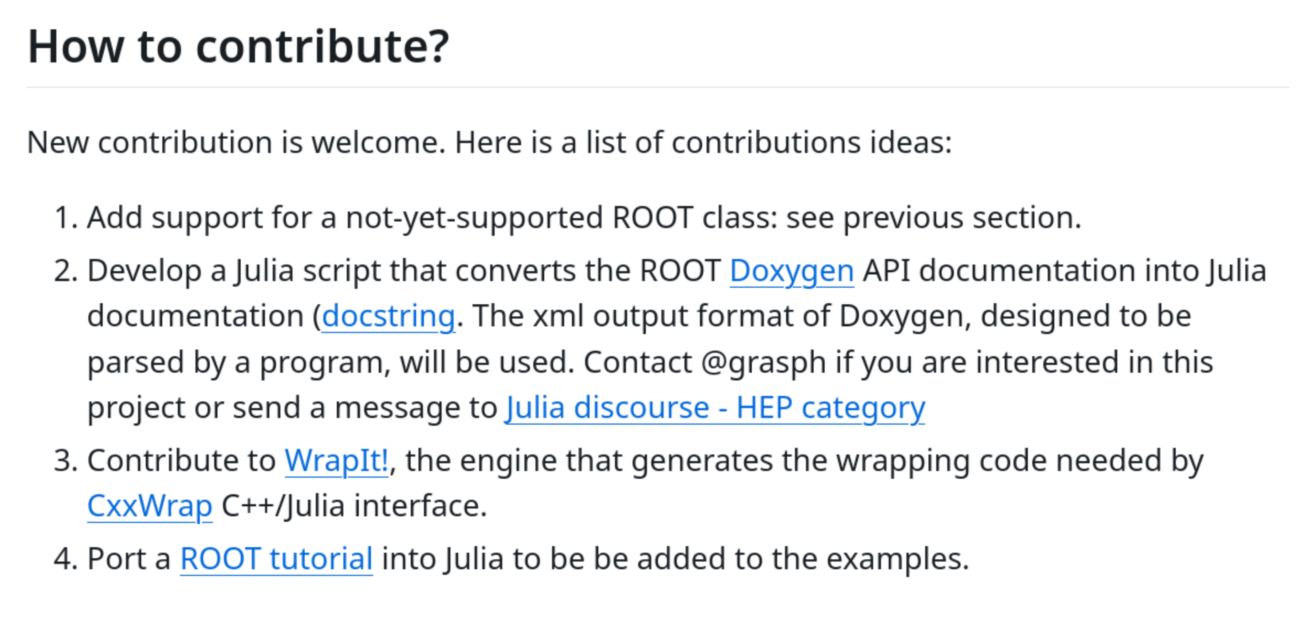
- Reading/Writing histograms is easy.
- Reading TTree is easy thanks to TTreeReader
- Writing TTree is difficult because of ”SetAddress” mechanism of ROOT.

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

Missing features

- Documentation: need to consult ROOT reference manual.
  - In addition, in method prototype, only argument type are transferred to Julia. Can be fixed thanks to a new CxxWrap feature.
- Support of more ROOT classes.
- Installation of ROOT libraries currently done with Conda.jl. Needs to move to _jll: cross-compilation compilation requirement is the show stopped.

How to contribute



How to contribute?

New contribution is welcome. Here is a list of contributions ideas:

1. Add support for a not-yet-supported ROOT class: see previous section.
2. Develop a Julia script that converts the ROOT Doxygen API documentation into Julia documentation (docstring. The xml output format of Doxygen, designed to be parsed by a program, will be used. Contact @grasph if you are interested in this project or send a message to Julia discourse - HEP category
3. Contribute to WrapIt!, the engine that generates the wrapping code needed by CxxWrap C++/Julia interface.
4. Port a ROOT tutorial into Julia to be be added to the examples.

From the project GitHub repository README

Summary

- WrapIt! which was developed as a proof-of-concept is now real tool.
  - Was also used to bring Geant4 to Julia
- ROOT.jl reimplemented with WrapIt! and CxxWrap to support nowadays Julia releases.
- Several places to contribute to the project.