Automatic generation of Julia bindings to libraries written C++

Philippe Gras

IRFU, CEA, Université Paris-Saclay, France

Nov 6-9, 2023

The context

- The high-performance / high-level duality of Julia makes it the ideal language for HEP.
- **Interface to our legacy code, mainly in C++, is essential to adopt Julia.**

The WrapIt project

Demonstrate Julia bindings can be provided to large C++ libraries in an automatised manner and with little maintenance effort.

Requirements on the desired bindings

- 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.

CxxWrap

The project is based on CxxWrap.

How to add Julia bindings to a C++ library with CxxWrap?

- Implement a shared library in C++, where you declare the types and functions to bind: the glueing code.

Behind the scene

- The shared library wraps the C++ functions/methods in C functions in order to use the Julia built-in C call.
- The Julia part of CxxWrap generates the Julia wrapper functions.

Example of glueing code to write to use CxxWrap

Example

```
//Create a binding for class A type:
auto t0 = types.add_type<A>("A");

//Create a binding for the method say_hello of class A:
t0.method("say_hello", &A::say_hello);
```

Note: *Additional lines of code can be needed in more sophisticated example.*

The code is simple, but can be cumbersome to write and maintain for large libraries.

WrapIt! Automatic generation of the glueing code

Developed the WrapIt! tool that generates the glueing code.

- Produces the glueing code from the library header files.
- Requires minimal configuration.

Challenges

- Interpreting content written in sophisticated language (C++20 standard: 1853 pages!).
- Header files ≠ API description.

Design choices

- Written in C++.
- Use of LLVM/CLANG:
  - mainly libclang: stable C API of clang libraries;
  - few calls to Clang AST C++ library for few missing features of libclang.

WrapIt! features

- Extracts the list of classes and functions to wrap from a provided set of **header files**.
- The list can be fine-tuned by providing an **exclusion list**.
- **Adds** to this list all **types** required to use the bound functions (argument and return types) and missing from the list.
- Maps **inheritance**: max. one parent class. Selection of parent class configurable.
- Optionally generates **accessors** for class/struct public fields.

WrapIt! demo

We will use as C++ library example, ROOT

- ROOT developed to **analyse** the **petabytes of data** produced at the LHC.
- Includes **all the tools used in HEP data analyses**: histogram, fit, machine learning, data unfolding, plotting, etc.
- **A large library**: exports more than 8000 C++ classes.

Presentation break for the demonstration

Demo based on ex002-Hello and ex002-ROOT

Limits of the current WrapIt! code

- Partial support of class **templates**, no yet support for function templates.
  - Facing limits of libclang: miss the parsing of template specialisation abstract syntax tree (AST).
  - Need to use the "LibTooling'' C++ interface of clang instead of its libclang interface or having more features in libclang.
- **Inheritance mapping** limited to one parent
  - Should be easy to extend WrapIt! in order to generate wrappers for all methods inherited from any parents;
  - *The mapped type relationship will remain a single inheritance as it's a constraint from the Julia language.*

Summary

- Prototype developed to test automatic generation of Julia bindings for C++ based on CxxWrap.
- It demonstrates the feasibility of such automation.
- Well advanced. More development needed to leverage the prototype to a production tool with full support of C++ templates.
- Used to provide a Julia interface to Geant4! See Pere Mato's Talk on Thursday.