Cling Documentation

Release 1.0~dev

The Cling Team

Aug 24, 2023

CONTENTS

1 Table of Contents **3**

1.1 When and why was Cling developed? . . . 3
1.2 Interactivity in C++ with Cling . . . 3
1.3 Why interpreting C++ with Cling? . . . 4
1.4 Used Technology . . . 5
1.5 Cling is (also, but not only) REPL . . . 5
1.6 Command Line . . . 6
1.7 Applications . . . 7
1.8 Conclusion . . . 8
1.9 Literature . . . 8

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ #include <string>
[cling]$ std::string s("abc");
[cling]$ s.find('b')
(std::basic_string<char, std::char_traits<char>, std::allocator<char> >::size_type) 1
[cling]$
```

Cling is an interactive C++ interpreter built on top of Clang and LLVM. It uses LLVM's *Just-In-Time* (JIT) compiler to provide a fast and optimized compilation pipeline. Cling uses the read-eval-print-loop (**REPL**) approach, making rapid application development in C++ possible, avoiding the classic edit-compile-run-debug cycle approach.

Cling's last release, download instructions, dependencies, and any other useful information for developers can be found on Cling's GitHub webpage.

Find out more about **Interpreting C++** on the Compiler Research Group's webpage.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 When and why was Cling developed?

Cling was first released in 2014 as the interactive, C++ interpreter in ROOT. ROOT is an open-source program written primarily in C++, developed by research groups in high-energy physics including CERN, FERMILAB and Princeton. ROOT is nowadays used by most high-energy physics experiments. CERN is an European research organization that operates the largest particle physics laboratory in the world. Its experiments collect petabytes of data per year to be serialized, analyzed, and visualized as C++ objects. In this framework, Cling was developed with the aim to facilitate the processing of scientific data in the field of high-energy physics . Cling is a core component of ROOT: it provides essential functionality for the analysis of vast amounts of very complex data produced by the experimental high-energy physics community by enabling (1) interactive exploration in C++, (2) dynamic interoperability (see cppyy, an automatic, runtime Python/C++ binder), and (3) rapid prototyping capabilities.

1.2 Interactivity in C++ with Cling

Interactive programming is a programming approach that allows developers to
change and modify the program as it runs. The final result is a program that actively responds to a developers' intuitions, allowing them to make changes in their code, and to see the result of these changes without interrupting the running program. Interactive programming gives programmers the freedom to explore different scenarios while developing software, writing one expression at a time, figuring out what to do next at each step, and enabling them to quickly identify and fix bugs whenever they arise. As an example, the High-Energy Physics community includes professionals with a variety of backgrounds, including physicists, nuclear engineers, and software engineers. Cling allows for interactive data analysis in ROOT by giving researchers a way to prototype their C++ code, allowing them to tailor it to the particular scope of the analysis they want to pursue on a particular set of data before being added to the main framework.

Interpreted language is a way to achieve interactive programming. In
statically compiled language, all source code is converted into native machine code and then executed by the processor before being run. An interpreted language instead runs through source programs line by line, taking an executable segment of source code, turning it into machine code, and then executing it. With this approach, when a change is made by the programmer, the interpreter will convey it without the need for the entire source code to be manually compiled. Interpreted languages are flexible, and offer features like dynamic typing and smaller program size.

Cling is not an interpreter, it is a Just-In-Time (JIT) compiler that feels
like an interpreter, and allows C++, a language designed to be compiled, to be interpreted. When using Cling, the programmer benefits from both the power of C++ language, such as high-performance,

robustness, fastness, efficiency, versatility, and the capability of an interpreter, which allows for interactive exploration and on-the-fly inspection of the source-code.

1.3 Why interpreting C++ with Cling?

1. **Learning C++:**

One use case of Cling is to aid the C++ learning process. Offering imediate feedback the user can easily get familiar with the structures and spelling of the language.

2. **Creating scripts:**

The power of an interpreter lays as well in the compactness and ease of repeatedly running a small snippet of code - aka a script. This can be done in Cling by inserting the bash-like style line:

```
#!/usr/bin/cling
```

3. **Rapid Application Development (RAD):**

Cling can be used successfully for Rapid Application Development allowing for prototyping and proofs of concept taking advantage of dynamicity and feedback during the implementation process.

4. **Runtime-Generated Code**

Sometime it's convenient to create code as a reaction to input (user/network/configuration). Runtime-generated code can interface with C++ libraries.

5. **Embedding Cling:**

The functionality of an application can be enriched by embedding Cling. To embed Cling, the main program has to be provided. One of the things this main program has to do is initialize the Cling interpreter. There are optional calls to pass command line arguments to Cling. Afterwards, you can call the interpreter from any anywhere within the application.

For compilation and linkage the application needs the path to the Clang and LLVM libraries and the invocation is order dependent since the linker cannot do backward searches.

```
g++ embedcling.cxx -std=c++11 -L/usr/local/lib
                  -lclingInterpreter -lclingUtils
                  -lclangFrontend -lclangSerialization -lclangParse -lclangSema
                  -lclangAnalysis -lclangEdit -lclangLex -lclangDriver -
↪lclangCodeGen
                  -lclangBasic  -lclangAST
                  `llvm-config
                    --libs bitwriter mcjit orcjit native option
                      ipo profiledata instrumentation objcarcopts`
                    -lz -pthread -ldl -ltinfo
                  -o embedcling
```

Embedding Cling requires the creation of the interpreter. Optionally compiler arguments and the resource directory of LLVM can be passed. An example is the following:

```
#include "cling/Interpreter/Interpreter.h"

int main(int argc, char** argv) {
   const char* LLVMRESDIR = "/usr/local/"; //path to llvm resource directory
   cling::Interpreter interp(argc, argv, LLVMRESDIR);
```

(continues on next page)

(continued from previous page)

```
  interp.declare("int p=0;");
}
```

A more complete example could be found in tools/demo/cling-demo.cpp.

1.4 Used Technology

LLVM is a free, open-source compiler infrastructure under the Apache License 2.0. It is designed as a collection of tools including Front Ends parsers, Middle Ends optimizers, and Back Ends to produce machine code out of those programs.

Clang is a front-end that uses a LLVM license. Clang works by taking the source language (e.g. C++) and translating it into an intermediate representation that is then received by the compiler back end (i.e., the LLVM backend). Its library-based architecture makes it relatively easy to adapt Clang and build new tools based on it. Cling inherits a number of features from LLVM and Clang, such as: fast compiling and low memory use, efficient C++ parsing, extremely clear and concise diagnostics, Just-In-Time compilation, pluggable optimizers, and support for GCC extensions.

Interpreters allow for exploration of software development at the rate of human thought. Nevertheless, interpreter code can be slower than compiled code due to the fact that translating code at run time adds to the overhead and therefore causes the execution speed to be slower. This issue is overcome by exploiting the *Just-In-Time* (JIT) compilation method, which allows an efficient meory management (for example, by evaluating whether a certain part of the source code is executed often, and then compile this part, therefore reducing the overall execution time).

With the JIT approach, the developer types the code in Cling's command prompt. The input code is then lowered to Clang, where is compiled and eventually transformed in order to attach specific behavior. Clang compiles then the input into an AST representation, that is then lowered to LLVM IR, an intermediate language that is not understood by the computer. LLVM's just-in-time compilation infrastructure translates then the intermediate code into machine language (eg. Intel x86 or NVPTX) when required for use. Cling's JIT compiler relies on LLVM's project ORC (On Request Compilation) Application Programming Interfaces (APIs).

1.5 Cling is (also, but not only) REPL

A read-eval-print-loop (**REPL**) is an interactive programming environment that takes user inputs, executes them, and returns the result to the user. In order to enable interactivity in C++, Cling provides several extensions to the C++ language:

1. **Defining functions in the global scope:** Cling redefines expressions at a global level. C++ provides limited support for this, Cling possesses the necessary semantics to re-define code while the program is running, minimizing the impedance mismatch between the **REPL** and the C++ codebase, and allowing for a seamlessly interactive programing experience.
2. **Allows for implementation of commands** that provide information about the current state of the environment. e.g., has an Application Programming Interface (**API**) to provide information about the current state of the environment.
3. **Error recovery:** Cling has an efficient error recovery system which allows it to handle the errors made by the user without restarting or having to redo everything from the beginning.

4. **Tight feedback loop:** It provides feedback about the results of the developer's choices that is both accurate and fast.
5. **Facilitates debugging:** The programmer can inspect the printed result before deciding what expression to provide for the next line of code.

1.6 Command Line

Cling has its own command line, which looks like any other Unix shell. The emacs-like command line editor is what we call interactive command line or interactive shell.

Once we start Cling it automatically includes several header files and its own runtime universe. Thus it creates the minimal environment for the user to start.

1.6.1 Grammar

Cling is capable to parse everything that Clang can do. In addition, Cling can parse some interpreter-specific C++ extensions.

1.6.2 Metaprocessor

Cling Metaprocessor provides convenient and easy to use interface for changing the interpreter's internal state or for executing handy commands. Cling provides the following metaprocessor commands:

syntax: .(command), where command is:

```
x filename.cxx
```

loads filename and calls void filename() if defined;

```
L library | filename.cxx
```

loads library or filename.cxx;

```
printAST
```

(DEBUG ONLY) shows the abstract syntax tree after each processed entity;

```
I path
```

adds an include path;

```
.@
```

Cancels the multiline input;

```
.dynamicExtensions
```

Turns on cling's dynamic extensions. This in turn enables the dynamic lookup and the late resolving of the identifier. With that option cling tries to heal the compile-time failed lookups at runtime.

1.7 Applications

1. **C++ in Jupyter Notebook - Xeus Cling:**

The Jupyter Notebook technology allows users to create and share documents that contain live code, equations, visualizations and narrative text. It enables data scientists to easily exchange ideas or collaborate by sharing their analyses in a straight-forward and reproducible way. Jupyter's official C++ kernel(Xeus-Cling) relies on Xeus, a C++ implementation of the kernel protocol, and Cling. Using C++ in the Jupyter environment yields a different experience to C++ users. For example, Jupyter's visualization system can be used to render rich content such as images, therefore bringing more interactivity into the Jupyter's world. You can find more information on Xeus Cling's Read the Docs webpage.

2. **Interactive CUDA C++ with Cling:**

CUDA is a platform and Application Programming Interface (API) created by NVIDIA. It controls GPU (Graphical Processing Unit) for parallel programming, enabling developers to harness the power of graphic processing units (GPUs) to speed up applications. As an example, PIConGPU is a CUDA-based plasma physics application to solve the dynamics of a plasma by computing the motion of electrons and ions in the plasma field. Interactive GPU programming was made possible by extending Cling functionality to compile CUDA C++ code. The new Cling-CUDA C++ can be used on Jupyter Notebook platform, and enables big, interactive simulation with GPUs, easy GPU development and debugging, and effective GPU programming learning.

3. **Clad:**

Clad enables automatic differentiation (AD) for C++. It was first developed as a plugin for Cling, and is now a plugin for Clang compiler. Clad is based on source code transformation. Given C++ source code of a mathematical function, it can automatically generate C++ code for computing derivatives of the function. It supports both forward-mode and reverse-mode AD.

4. **Cling for live coding music and musical instruments:**

The artistic live coding community has been growing steadily since around the year 2000. The Temporary Organisation for the Permanence of Live Art Programming (TOPLAP) has been around since 2004, Algorave (algorithmic rave parties) recently celebrated its tenth birthday, and six editions of the International Conference on Live Coding (ICLC) have been held. A great many live coding systems have been developed during this time, many of them exhibiting exotic and culturally specific features that professional software developers are mostly unaware of. In this framework, Cling has been used as the basis for a C++ based live coding synthesiser (TinySpec-Cling). In another example, Cling has been installed on a BeagleBoard to bring live coding to the Bela interactive audio platform (Using the Cling C++ Interpreter on the Bela Platform). These two examples show the potential mutual benefits for increased engagement between the Cling community and the artistic live coding community.

5. **Clion:** The CLion platform is a Integrating Development Environment (IDE) for C and C++ by JetBrains. It was developed with the aim to enhance developer's productivity with a smart editor, code quality assurance, automated refactorings and deep integration with the CMake build system. CLion integrates Cling, which can be found by clicking on Tool. Cling enables prototyping and learning C++ in CLion. You can find more information on CLion's building instructions.

1.8 Conclusion

Cling is not just an interpreter, and is not just a REPL: it is a C/C++ JIT-compiler that can be embedded to your software for efficient incremental execution of C++. Cling allows you to decide how much you want to compile statically and how much to defer for the target platform. Cling enables reflection and introspection information in high-performance systems such as ROOT, or Xeus Jupyter, where it provides efficient code for performance-critical tasks where hot-spot regions can be annotated with specific optimization levels. You ca find more information regarding Cling's internal architecture, functionment, user-cases, and Cling's based project into the References Chapter.

1.9 Literature

Table 1: What is Cling?

Link	Info	Description
Relaxing the One Definition Rule in Interpreted C++	*Javier Lopez Gomez et al.* 29th International Conference on Compiler Construction 2020	This paper discusses how Cling enables redefinitions of C++ entities at the prompt, and the implications of interpreting C++ and the One Definition Rule (ODR) in C++
Cling – The New Interactive Interpreter for ROOT 6	*V Vasilev et al* 2012 J. Phys.: Conf. Ser. 396 052071	This paper describes the link between Cling and ROOT. The concepts of REPL and JIT compilation. Cling's methods for handling errors, expression evaluation, streaming out of execution results, runtime dynamism.
Interactive, Introspected C++ at CERN	*V Vasilev*, CERN PH-SFT, 2013	Vassil Vasilev (Princeton University) explains how Cling enables interactivity in C++, and illustrates the type introspection mechanism provided by the interpreter.
Introducing Cling, a C++ Interpreter Based on Clang/LLVM	*Axel Naumann* 2012 Googletechtalks	Axel Naumann (CERN) discusses Cling's most relevant features: abstract syntax tree (AST) production, wrapped functions, global initialization of a function, delay expression evaluation at runtime, and dynamic scopes.
Creating Cling, an interactive interpreter interface	*Axel Naumann* 2010 LLVM Developers' meeting	This presentation introduces Cling, an ahead-of-time compiler that extends C++ for ease of use as an interpreter.

Table 2: Demos, tutorials, Cling's ecosystem:

Link	Info	Description
Cling integration | CLion	2022.2 Version	CLion uses Cling to integrate the Quick Documentation popup by allowing you to view the value of the expressions evaluated at compile time.
Interactive C++ for Data Science	*Vassil Vassilev* 2021 CppCon (The C++ Conference)	In this video, the author discusses how Cling enables interactive C++ for Data Science projects.
Cling – Beyond Just Interpreting C++	*Vassil Vassilev* 2021 The LLVM Project Blog	This blog page discusses how Cling enables template Instantiation on demand, language interoperability on demand, interpreter/compiler as a service, plugins extension.
TinySpec-Cling	Noah Weninger 2020	A tiny C++ live-coded overlap-add (re)synthesizer for Linux, which uses Cling to add REPL-like functionality for C++ code.
Interactive C++ for Data Science	*Vassil Vassilev, David Lange, Simeon Ehrig, Sylvain Corlay* 2020 The LLVM Project Blog	Cling enables eval-style programming for Data Science applications. Examples of ROOT and Xeus-Cling for data science are shown.
Interactive C++ with Cling	*Vassil Vassilev* 2020 The LLVM Project Blog	This blog page briefly discusses the concept of interactive C++ by presenting Cling's main features, such as wrapper functions, entity redefinition, error recovery.
Using the Cling C++ Interpreter on the Bela Platform	Jack Armitage 2019	Cling has been installed on a BeagleBoard to bring live coding to the Bela interactive audio platform.
Implementation of GlobalModuleIndex in ROOT and Cling	*Arpitha Raghunandan* 2012 Google Summer of Code GSoC	GlobalModuleIndex can be used for improving ROOT's and Cling's performance
Example project using cling as library	*Axel Naumann* 2016 GitHub	This video showcases how to use Cling as a library, and shows how to set up a simple CMake configuration that uses Cling.
Cling C++ interpreter testdrive	*Mika* 2015 Youtube	In this tutorial, a developer tries Cling for the first time by uploading a few simple C++ user-cases onto Cling, involving also the loading of external files
Building an Order Book in C++	*Dimitri Nesteruk* 2015 Youtube	This demo shows how to build a simple order book using C++, CLion, Google Test and, of course, Cling.
Cling C++ interpreter testdrive	Dimitri Nesteruk 2015 Youtube	This tutorial describes Cling's general features. You will learn how to start Cling on Ubuntu, how to write a simple expression (N=5, N++) and how to define a Class for calculating body mass index.
Cling Interactive OpenGL Demo	*Alexander Penev* 2012 Youtube	This demo shows how to use Cling for interactive OpenGL. A rotating triangle with changing color, a static figure, and a figure with light effects are created.

Table 3: Language Interoperability with Cling:

Link	Info	Description
Compiler Research - Calling C++ libraries from a D-written DSL: A cling/cppyy-based approach	*Alexandru Militaru* 2021 Compiler-Research Meeting	This video presents D and C++ interoperability through SIL-Cling architecture

Table 4: Interactive CUDA C++ with Cling:

Link	Info	Description
Adding CUDA® Support to Cling: JIT Compile to GPUs	*Simeon Ehrig* 2020 LLVM Developer Meeting	Interactive CUDA-C++ through Cling is presented. Cling-CUDA architecture is discussed in detail, and an example of interactive simulation for laser plasma applications is shown.

Table 5: C++ in Jupyter Notebook - Xeus Cling:

Link	Info	Description
Interactive C++ code development using C++Explorer and GitHub Classroom for educational purposes	*Patrick Diehl* 2020 Youtube	C++Explorer is a novel teaching environment based on Jupyterhub and Cling, adapted to teaching C++ programming and source code management.
Deep dive into the Xeus-based Cling kernel for Jupyter	*Vassil Vassilev* 2021 Youtube	Xeus-Cling is a Cling-based notebook kernel which delivers interactive C++.
Xeus-Cling: Run C++ code in Jupyter Notebook	*LearnOpenCV* 2019 Youtube	In this demo, you will learn an example of C++ code in Jupyter Notebook using Xeus-Cling kernel.

Table 6: Clad:

Link	Info	Description
Clad: Automatic differentiation plugin for C++	Read The Docs webpage	Clad is a plugin for Cling. It allows to perform Automatic Differentiation (AD) on multivariate functions and functor objects

Note: This project is under active development. Cling has its documentation hosted on Read the Docs.