Gene Weber – June 2022
Linux & GCC specific
Cling Version 1.0~dev

# Cling Overview

“Cling is an interactive C++ interpreter, built on the top of LLVM and Clang libraries. Its advantages over the standard interpreters are that it has a command line prompt and uses just-in-time (JIT) compiler for compilation.” - https://root.cern/cling/

`$ Cling --help` returns the command line options for Cling and Clang.

Calling Cling without a cpp file invokes it in interactive mode and brings up a Cling terminal prompt, `[cling]$.`

- Entering `.?` at the prompt provides help for the available interactive commands.
- Entering `.q` exits the interactive session.
- Interactive Cling creates a `.cling_history` file in the `$HOME` directory.

## Interactive and non-interactive modes

As noted above, if Cling is invoked without a cpp file, it executes in interactive mode. If the default header is enabled, the interactive session looks like this.

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$
```

This interface can be used as a C++ interpreter.

```
[cling]$ #include <iostream>
[cling]$ std::cout << "Hello World!\n";
Hello World!
[cling]$
```

The `.x` command executes a function from a file. The file name must be consistent with the function name. The file may contain multiple functions, but the file name must match the function to be called. For example the following could be the contents of a file named `hello.cpp`.

```
#include <iostream>

int ret_num() {
      return 1;
}

void hello() {
    std::cout << ret_num() << " Hello World!\n";
}
```

This function can now be executed at the cling prompt as follows:

```
[cling]$ .x hello.cpp
1 Hello World!
[cling]$
```

To execute a C++ file in non-interactive mode, the file must have `#! filename` as the first line, and the function must be called in the file to take effect. The `hello.cpp` file above would be modified as follows:

```
#! hello.cpp
#include <iostream>

int ret_num() {
     return 1;
}

void hello() {
    std::cout << ret_num() << " Hello World!\n";
}

hello();
```

It can now be run in non-interactive mode by including it on the Cling command line.

```
$ cling hello.cpp
1 Hello World!
$
```

# Creating and using shared libraries with Cling

Compiling C++ code into shared libraries that are called in Cling can improve run time performance. The C++ code must have a header file containing definitions of all functions and subroutines.

Compile your library source code into object code.

```
g++ -fPIC -c -Wall -pedantic your_library.cpp
```

This will generate *your_library*.o

Compile object code into a shared library.

```
g++ -shared your_library.o -o libyour_library.so
```

Example:
file: example.h

```
void prnt_num(int num);
```

```
int sq_num(int num);
```

file: example.cpp

```
#include "example.h"
#include <iostream>

void prnt_num(int num) {
  std::cout << "Number = " << num << ".\n";
}

int sq_num(int num) {
    return num * num;
}
```

Compile the shared library:

```
g++ -fPIC -c -Wall -pedantic example.cpp
g++ -shared example.o -o libexample.so
```

**-fPIC** = Generate Position Independent Code. Code is relocatable in memory.
**-c** = Translate source code to object code.
**-Wall** = Warn All. Warnings that "all" agree upon.
**-pedantic** = The C and C++ standards specify that certain language extensions are forbidden. This option causes GCC to issue warnings when such cases are encountered.
**-shared** = Create a shared library.
**-o** = Specify the name of the output file.

In Cling, the `.L` command loads the given file or library. The header file is included to declare library functions and subroutines.

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ .L libexample.so
[cling]$ #include "example.h"
[cling]$ prnt_num(8);
Number = 8.
[cling]$ prnt_num(sq_num(8));
Number = 64.
[cling]$
```

Libraries and include files can also be specified during Cling invocation. Their specification is similar to specifying them when invoking a C++ compiler.

```
cling -L/library_directory_path -llibrary_file \
-I/include_directory_path -include include_file
```

For this example Cling could be invoked as follows:

```
$ cling -L/home/cling -lexample -I/home/cling -include example.h
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
```

```
****************************************
[cling]$ prnt_num(8);
Number = 8.
[cling]$ prnt_num(sq_num(8));
Number = 64.
[cling]$
```

# How to embed Cling in another C++ program

One way to embed Cling into another C++ program is to convert Cling's `main()` function into a function called by the new C++ program. The file containing Cling's main call is `cling.cpp` located in the `src/tools/cling/tools/driver` directory of the Cling installation.

## Compilation

A compilation script for `cling.cpp` can be generated as follows. Update the file timestamp by using `touch`.

```
touch src/tools/cling/tools/driver/cling.cpp
```

Rerun `make` with the verbose option set to 1 to capture all of the commands that `make` is executing.

```
cd obj
make VERBOSE=1 -j 8 2>&1 | tee compile.sh
mv compile.sh your_directory
```

This line specifies 8 jobs in parallel because the node used in this example has 8 cores. It redirects std error into std out, and captures both in the file `compile.sh`.

The following are my recommended edits to compile.sh. I have included a bash script listing at the end of this document that automates these changes.

- Delete all lines except the two executing compilation. Put a blank line between these two lines. In my case the two lines containing `/usr/bin/c++`.
- Since you will be compiling in your local working directory, delete any `cd /path` in front of `/usr/bin/c++`.
- Insert `#! /bin/bash` as the first line in the file.
- Insert `INSTALL_DIR=/absolute/path/to/your/cling/installation` as the second line in the file.
- Change all relative paths to absolute using `$INSTALL_DIR`. You can also change any absolute paths to the installation directory with `$INSTALL_DIR`.
- Replace spaces on the two compile lines with a line continuation character " \" and a carriage return. This may require some manual fixes. Example:

  ```
  usr/bin/c++ -fPIC -fvisibility-inlines-hidden -Werror=date-time ...
  ```

  Becomes:

  ```
  /usr/bin/c++ \
  -fPIC \
  -fvisibility-inlines-hidden \
  ```

```
-Werror=date-time \
```

- Replace
  ```
  -o \
  CMakeFiles/cling.dir/cling.cpp.o \
  ```
  With
  ```
  -o embed_cling.cpp.o \
  ```
- Replace
  ```
  -c \
  $INSTALL_DIR/src/tools/cling/tools/driver/cling.cpp
  ```
  With
  ```
  -c embed_cling.cpp
  ```
- Replace
  ```
  CMakeFiles/cling.dir/cling.cpp.o \
  ```
  With
  ```
  embed_cling.cpp.o \
  ```
- Replace
  ```
  -o \
  $INSTALL_DIR/obj/lib/bin/cling \
  ```
  With
  ```
  -o embed_cling.cpp \
  ```
- Delete the line containing:
  ```
  \$ORIGIN/../lib
  ```
- I replaced `/usr/bin/c++` with g++.
- I added the comment line `# Compile object` before the first compile line, and `# Compile executable` before the second compile line.

Now make a copy of `cling.ccp` in your working directory as embed_cling which is the name used in the edits above.

```
cp src/tools/cling/tools/driver/cling.cpp embed_cling.cpp
```

The following edits should be done to `embed_cling.cpp` to make it a program that embeds Cling. First, `main( int argc, char **argv )` needs to be changed to a function name such as `run_cling(int argc, char **argv)`. Second, a minimal main function that calls the Cling function must now be added. For example:

```
int main( int argc, char **argv ) {
    int status = run_cling(argc, argv);
    return status;
}
```

The Cling interpreter needs to access the “llvm resource directory”. This is the directory that contains `libclang.so.9`. In the Cling installation it is `obj/lib`. The Cling interpreter is looking for the location of the `lib` directory. When invoked as stand alone Cling, the interpreter gets the correct default value for the director location. When embedding Cling, pass it explicitly to the interpreter by changing:

```
cling::Interpreter Interp(argc, argv);
```

To:

```
const char* LLVMRESDIR = "/home/ubuntu/software/cling/obj";
cling::Interpreter Interp(argc, argv, LLVMRESDIR);
```

Failing to do this will result in the following run time error when the embedded program is invoked.

```
ERROR in cling::CIFactory::createCI():
  resource directory /home/software/cling/lib/clang/9.0.1 not found!
In file included from input_line_1:1:
In file included from /usr/include/c++/9/new:40:
In file included from /usr/include/c++/9/exception:143:
In file included from /usr/include/c++/9/bits/exception_ptr.h:38:
/usr/include/c++/9/bits/cxxabi_init_exception.h:38:10: fatal error:
'stddef.h' file not found
#include <stddef.h>
         ^~~~~~~~~~
Replaced symbol atexit cannot be found in JIT!
Replaced symbol at_quick_exit cannot be found in JIT!
```

Linux users can scrap the “if defines” for WIN32.

Obviously this is an extremely trivial C++ program that does no more than call Cling. Any meaningful development won’t have `main()` in this file. And of course the essence of the compile script created here will need to be integrated into the compile scripting for the larger C++ development.

I hope you found this introductory document helpful.

make_comp_script listing:

```
#! /bin/bash
# June 2022 - Gene Weber
# Convert VERBOSE make log into compile script.

if [ "$#" -ne 4 ]; then
    echo "make_comp_script  input_file  output_file  new_C++_file_name
cling_installation_ directory"
    exit 1
fi

NAME_ROOT=`awk -F'.' '{print $1}' <<< $3`

grep "c++" $1 | \
sed s/\ $// | \
sed s/^cd\ .*c++\ /\#\\ng++\ / | \
sed s/^.*c++\ /\#\#\\ng++\ / | \
sed s/\ \ */\ \\n/g | \
sed '/^-o/N;s/\n//' | \
sed '/^-c/N;s/\n//' | \
sed '/^-lm/N;s/\n//' | \
sed s/CMakeFiles.*cling\.cpp/$3/ | \
sed s/^-c\ .*cling\.cpp/-c\ $3/ | \
sed s/^-o\ .*cling/-o\ $NAME_ROOT/ | \
sed "s|^\.\./\.\./\.\./\.\.|\$INSTALL_DIR/obj|" | \
sed "s|^\.\./\.\.|\$INSTALL_DIR/obj/tools/cling|" | \
sed "s|$4|\$INSTALL_DIR|" | \
```

```
sed '/ORIGIN/d'  | \
sed 's/\ $/\ \\/' | \
sed "s|^##|\\n# Compile executable|" | \
sed "s|^#$|#! /bin/bash\\nINSTALL_DIR=$4\\n\\n# Compile object|" >$2

chmod 755 $2
```