Gradle Guides (https://guides.gradle.org)

Building and Testing C++ Libraries

Contents

This guide shows how to

1. Build a simple C++ library using the Gradle `cpp` plugin, and
2. Build and Run unit tests written with `GoogleTest` using the `google-test` plugin.

What you'll need

- About 14 minutes
- A text editor
- A command prompt
- The Java Development Kit (JDK), version 1.7 or higher
- A Gradle distribution (https://gradle.org/install), version 4.10-rc-2 or better
- An installed C++ compiler. See which C++ tool chains (https://docs.gradle.org/4.10-rc-2/userguide/native_software.html#native-binaries:tool-chain-support) are supported by Gradle and whether you have to do any installation configuration (https://docs.gradle.org/4.10-rc-2/userguide/native_software.html#sec:tool_chain_installation) for your platform.
- GoogleTest static library and the header files

Layout

The first step is to create a folder for the new project and add a Gradle Wrapper (https://docs.gradle.org/4.10-rc-2/userguide/gradle_wrapper.html#sec:wrapper_generation) to the project.

```
$ mkdir building-and-testing-cpp-libraries
$ cd building-and-testing-cpp-libraries
$ gradle wrapper  1

:wrapper

BUILD SUCCESSFUL
```

1 This allows a version of Gradle to be locked to a project and henceforth you can use `./gradlew` instead of `gradle`.

Create a `build.gradle` file that has the following content:

build.gradle

GROOVY

```
apply plugin: 'cpp'  1
apply plugin: 'google-test'  2

model {  3
    repositories {
        libs(PrebuiltLibraries) {  4
            googleTest {  5
                headers.srcDir new File(projectDir, "3rdparty/googletest-master/googletest/inc
                binaries.withType(StaticLibraryBinary) {
                    staticLibraryFile = file(projectDir.getPath() + "/3rdparty/googletest-mast
                }
            }
        }
    }
    components {
        greeter(NativeLibrarySpec) { }  8
        // Let's try using the library
        main(NativeExecutableSpec) {  9
            sources {
                cpp.lib library: "greeter"  10
            }
        }
    }


    binaries {
        withType(SharedLibraryBinarySpec) {
            if (toolChain in VisualCpp) {
                cppCompiler.define "DLL_EXPORT"  11
            }
        }
        withType(GoogleTestTestSuiteBinarySpec) {
            lib library: "googleTest", linkage: "static"  12

            if (targetPlatform.operatingSystem.linux) {
                cppCompiler.args '-pthread'
                linker.args '-pthread'
            }
        }
    }
}

def findGoogleTestStaticLib(Platform p) {
    if (p.operatingSystem.windows) {
        return "Release/gtest.lib"
    } else {
        return "libgtest.a"
    }
}
```

1 C++ libraries and binaries are built using the `cpp` plugin.

2 Googletest based unit test binaries are built using the `google-test` plugin

3. Native builds use the `model` block
4. You should already have the static library for `GoogleTest`. Hence this is defined as `PrebuiltLibraries`.
5. The name of the library component is `googleTest`. This can be any meaningful name you choose.
6. Specify the path where the header files of GoogleTest is located. Test suites will include the header files from this location.
7. Specify the path of the static library. The file name depends on the platform. So, a utility function that returns the library filename based on the target platform is used.
8. Native libraries are specified as a component of type NativeLibrarySpec (https://docs.gradle.org/4.10-rc-2/dsl/org.gradle.nativeplatform.NativeLibrarySpec.html) and defined by a name - `greeter` in this case.
9. Native executables are specified as a component of type NativeExecutableSpec (https://docs.gradle.org/4.10-rc-2/dsl/org.gradle.nativeplatform.NativeExecutableSpec.html) and defined by a name - `main` in this case.
10. The `main` binary uses the `greeter` {c++} library and is specified using `cpp.lib`
11. When using the VC++ compiler, the library methods should be declared with `__declspec(dllexport)` for shared library. This defines the macro `DLL_EXPORT` based on which the declaration is modified.
12. The test binaries should be linked with the `GoogleTest` static library to be able to run. We specify the `linkage` as static to tell the linker to statically link the binary with the `googleTest` library defined in step 5 above.

Now run

```
$ ./gradlew tasks
```

You should see a number of tasks that has been added by Gradle

```
greeterSharedLibrary - Assembles shared library 'greeter:sharedLibrary'.
greeterStaticLibrary - Assembles static library 'greeter:staticLibrary'.

Build Dependents tasks
----------------------
assembleDependentsGreeter - Assemble dependents of native library 'greeter'.
assembleDependentsGreeterSharedLibrary - Assemble dependents of shared library 'greeter:shared
assembleDependentsGreeterStaticLibrary - Assemble dependents of static library 'greeter:static
assembleDependentsGreeterTestGoogleTestExe - Assemble dependents of google test exe 'greeterTe
buildDependentsGreeter - Build dependents of native library 'greeter'.
buildDependentsGreeterSharedLibrary - Build dependents of shared library 'greeter:sharedLibrar
buildDependentsGreeterStaticLibrary - Build dependents of static library 'greeter:staticLibrar
buildDependentsGreeterTestGoogleTestExe - Build dependents of google test exe 'greeterTest:goo

Verification tasks
------------------
check - Runs all checks.
checkGreeterTestGoogleTestExe - Check google test exe 'greeterTest:googleTestExe'
```

There are a few things to note here

- The names `Greeter` and `Main` in the task names are derived from the names of the components defined in `build.gradle`.
- The `NativeLibrarySpec`, by default, generates both static and shared(dynamic) libraries. By default, the shared library is used by the binaries that use this library
- The `google-test` plugin automatically generates testsuites for each of the component declared as a NativeLibrarySpec or NativeExecutableSpec. The testsuites are of type GoogleTestSuiteBinarySpec
- The `google-test` plugin has also added a new *verification task*. The verification tasks are `check` tasks that assemble and test a binary.
- You might see a few more tasks than what has been listed above.

Add source code

In `build.gradle`, there are two components. `greeter` is the library you will be building and `main` is a simple executable to consume this library. The sources are structured as below

- `greeter` library - `src/greeter/cpp` for sources and *non-exported header files* and `src/greeter/headers` for *exported header files*
- `main` binary - `src/main/cpp` for sources.

The `google-test` plugin will automatically create two components named `greeterTest` and `mainTest` respectively for the `greeter` component and `main` component declared in te build scrpt. Place the test sources for the `greeterTest` test component at `src/greeterTest/cpp`.

In general, the sources are located at `src/{name}/cpp` where `name` is the name of the component.

```
mkdir -p src/greeter/cpp src/greeter/headers
mkdir -p src/main/cpp
mkdir -p src/greeterTest/cpp
```

Place a `greeter.hpp` in `src/greeter/headers` and a `greeter.cpp` in `src/greeter/cpp`.

src/greeter/headers/greeter.hpp

CPP

```
#ifndef _GREETER_H_
#define _GREETER_H_

#include <string>
#include <algorithm>

#if defined(DLL_EXPORT)
#define DECLSPEC __declspec(dllexport)
#else
#define DECLSPEC
#endif


class DECLSPEC Greeter {
public:
    Greeter(std::string name_) : name(name_) {};
    Greeter() : name("World") {};
    void greet();
    int getNameLength();
private:
    std::string name;
};

#endif
```

src/greeter/cpp/greeter.cpp

CPP

```
#include <iostream>
#include "greeter.hpp"

void Greeter::greet() {
    std::cout << "Hello, " << name << ", your name has " << getNameLength() << " chars." << st
}

int Greeter::getNameLength() {
    return name.length();
}
```

Now place a `greeting.cpp` under `src/main/cpp`.

src/main/cpp/greeting.cpp

CPP

```
#include "greeter.hpp"

int main(int argc, char* argv[]) {
    Greeter g("Gradle User");
    g.greet();
    return 0;
}
```

Place a `greetertest.cpp` under `src/greeterTest/cpp`.

src/greeterTest/cpp/greetertest.cpp

CPP

```
#include <gtest/gtest.h>
#include "greeter.hpp"

TEST(GreeterTest, CheckCapitalisation) {
    Greeter g("GradleUser");
    EXPECT_EQ(g.getNameLength(), 10);
}

int main(int argc, char **argv) {
  ::testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

Build the project

Gradle adds tasks to the build for both static and shared form of the library. To build just the static library, run

```
$ ./gradlew greeterStaticLibrary
:compileGreeterStaticLibraryGreeterCpp 1
:createGreeterStaticLibrary
:greeterStaticLibrary

BUILD SUCCESSFUL
```

1 Gradle does not show the compiler commands or only show an excerpt of the compiler output if the compilation failed in order to keep the output from Gradle itself clean. But the compiler output can be found in `build/tmp/compileGreeterStaticLibraryGreeterCpp/output.txt` and the compiler flags can be found in `build/tmp/compileGreeterStaticLibraryGreeterCpp/options.txt`

This builds the static library under `build/libs/greeter/static`

```
$ find build/libs/
build/libs/
build/libs/greeter
build/libs/greeter/static
build/libs/greeter/static/libgreeter.a
```

In a similar fashion, the shared library can be generated by running

```
$ ./gradlew greeterSharedLibrary
```

```
$ find build/libs/
build/libs/
build/libs/greeter
build/libs/greeter/shared
build/libs/greeter/shared/libgreeter.so
build/libs/greeter/static
build/libs/greeter/static/libgreeter.a
```

Since the `main` component depends on the `greeter` library, running the `mainExecutable` task will generate the static for exported symbols only and shared libraries and the executable binary as well.

```
$ ./gradlew mainExecutable
:compileMainExecutableMainCpp
:compileGreeterSharedLibraryGreeterCpp
:linkGreeterSharedLibrary
:greeterSharedLibrary
:linkMainExecutable
:mainExecutable

BUILD SUCCESSFUL
```

You can run the binary that uses this library (you need to set the LD_LIBRARY_PATH as needed so that the shared library can be located).

```
$ LD_LIBRARY_PATH=build/libs/greeter/shared/ ./build/exe/main/main
Hello, Gradle User, your name has 11 chars.
```

Test the project

As mentioned earlier, the `google-test` plugin automatically adds the testsuite components and verification tasks to the project. When you run the `build` task, gradle runs the compile tasks for source code and the test suites. It then runs the testsuite binary as well. The test binary in this case will be named `greeterTest` and the task will be named `runGreeterTestGoogleTestExe`.

```
$ ./gradlew build
:compileGreeterSharedLibraryGreeterCpp
:linkGreeterSharedLibrary
:greeterSharedLibrary
:compileGreeterStaticLibraryGreeterCpp
:createGreeterStaticLibrary
:greeterStaticLibrary
:compileMainExecutableMainCpp
:linkMainExecutable
:mainExecutable
:assemble
:checkGreeterSharedLibrary UP-TO-DATE
:compileGreeterTestGoogleTestExeGreeterCpp
:compileGreeterTestGoogleTestExeGreeterTestCpp
:linkGreeterTestGoogleTestExe
:greeterTestGoogleTestExe
:installGreeterTestGoogleTestExe
:runGreeterTestGoogleTestExe
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from GreeterTest
[ RUN      ] GreeterTest.CheckCapitalisation
[       OK ] GreeterTest.CheckCapitalisation (0 ms)
[----------] 1 test from GreeterTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
:checkGreeterTestGoogleTestExe
:checkGreeterStaticLibrary
:compileMainTestGoogleTestExeMainCpp
:linkMainTestGoogleTestExe
:mainTestGoogleTestExe
:installMainTestGoogleTestExe
:runMainTestGoogleTestExe
Hello, Gradle User, your name has 11 chars.
:checkMainTestGoogleTestExe
:checkMainExecutable
:check
:build
```

You should see the output from the `GoogleTest` testsuite that you had written for the `greeter` library as part of the output for the `build` task.

Summary

You have created a C++ library (both static and shared) and written unit tests for it. In doing so, you saw

- How to create gradle build scripts for C++ libraries.
- How to generate static and/or shared libraries from C++ soruces.

- How to use our library in a C++ binary.
- How to build and run testsuites written with GoogleTest

Help improve this guide

Have feedback or a question? Found a typo? Like all Gradle guides, help is just a GitHub issue away. Please add an issue or pull request to gradle-guides/building-cpp-libraries (https://github.com/gradle-guides/building-cpp-libraries/) and we'll get back to you.