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

Building C++ Executables

Schalk Cronjé

Table of Contents

What you'll build
What you'll need
Layout
Add source code
Build your project
Summary
Next Steps
Help improve this guide

What you'll build

This guide demonstrates how to create a minimalist C++ executable using Gradle's `cpp` plugin.

What you'll need

- About 6 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 3.5 or better
- An installed C++ compiler. See which C++ tool chains (https://docs.gradle.org/3.5/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/3.5/userguide/native_software.html#sec:tool_chain_installation) for your platform.

Layout

The first step is to create a folder for the new project and add a Gradle Wrapper (https://docs.gradle.org/3.5/userguide/gradle_wrapper.html#sec:wrapper_generation) to the project.

```
$ mkdir cpp-executable
$ cd cpp-executable
$ 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 minimalist `build.gradle` file with the following content:

build.gradle

GROOVY

```
apply plugin : 'cpp'  1

model {  2
    components {
        main(NativeExecutableSpec)   3   4
    }
}
```

1. C++ projects are enabled via the `cpp` plugin
2. All native build definitions are done within a `model` block.
3. A native executable component is defined by a name - `main` in this case. This will determine the default location of source code, as well as the final name of the executable.
4. An executable is specified by using NativeExecutableSpec (https://docs.gradle.org/3.5/dsl/org.gradle.nativeplatform.NativeExecutableSpec.html).

If you run

```
$ ./gradlew tasks
```

you will see in the output that Gradle has added a number of tasks

```
installMainExecutable - Installs a development image of executable 'main:executable'
mainExecutable - Assembles executable 'main:executable'.

Build Dependents tasks
----------------------
assembleDependentsMain - Assemble dependents of native executable 'main'.
assembleDependentsMainExecutable - Assemble dependents of executable 'main:executable'.
buildDependentsMain - Build dependents of native executable 'main'.
buildDependentsMainExecutable - Build dependents of executable 'main:executable'.
```

Note the use of `Main` in the task names which are a direct deriviative of the component being called `main`.

Add source code

By convention, C++ projects in Gradle will follow a more contemporary layout. This can be troublesome for you if you are used to building C++ code with build tools that do not use a convention-over-configuration approach. Rest assured that Gradle is very configurable in this regard and should you need to migrate a C++ project to Gradle you can consult the C++ sources (https://docs.gradle.org/3.5/userguide/native_software.html#sec:cpp_sources) section of the User Guide.

In the `build.gradle` you have previsouly defined the executable component to be called `main`. By convention, this will means that Gradle will look in `src/main/cpp` for source files and non-exported header files. Create this folder

```
$ mkdir -p src/main/cpp
```

and place a `main.cpp` an a `greeting.hpp` within.

src/main/cpp/main.cpp

CPP

```
#include <iostream>   1
#include "greeting.hpp"  2

int main(int argc, char** argv) {
    std::cout << greeting << std::endl;
    return 0;
}
```

1 The standard C++ headers wil be located via the compiler toolchain that Gradle uses

2 Non-exported headers will be searched for relative to the specified C++ source folders. (In this case `src/main/cpp`).

src/main/cpp/greeting.hpp

CPP

```
#ifndef GRADLE_GUIDE_EXAMPLE_GREETING_HPP__
#define GRADLE_GUIDE_EXAMPLE_GREETING_HPP__

namespace {
    const char * greeting = "Hello, World";
}

#endif
```

Build your project

To build your project you can simply do `./gradlew build` as per usual, but if you specifically want to build the executable, run the task that Gradle has created for you:

```
$ ./gradlew mainExecutable

:compileMainExecutableMainCpp  1
:linkMainExecutable  2
:mainExecutable

BUILD SUCCESSFUL
```

1. To keep things tidy on the console, Gradle does not display compiler output. If you need to ever diagnose a compilation issue, the output from the compiler is stored in `build/tmp/compileMainExecutableMainCpp/output.txt`.
2. In a similar fashion the output from the linker is written to `build/tmp/linkMainExecutable/output.txt`

Look inside the `build` folder and you will notice the appearance of a `exe` folder. By convention Gradle will place all executables in subfolders named according to the component name. In this case you will find your assembled executable in `build/exe/main` and it will be called `main` (or `main.exe` under Windows).

Now run your newly built executable.

```
$ ./build/exe/main/main

Hello World
```

Congratulations! You have just built a C++ executable with Gradle.

Summary

You have created an C++ project that can be used as a foundation for something more substantial. In the process you saw:

- How to create a build script for C++ executables.

- Where to add source code by convention.
- How to build the executable without building the full project.

Next Steps

- Testing using CUnit (http://cunit.sourceforge.net) or GoogleTest (https://github.com/google/googletest) is supported. Gradle will respectively create a matching CUnitTestSuiteSpec (https://docs.gradle.org/3.5/dsl/org.gradle.nativeplatform.test.cunit.CUnitTestSuiteSpec.html) or GoogleTestTestSuiteSpec (https://docs.gradle.org/3.5/dsl/org.gradle.nativeplatform.test.googletest.GoogleTestTestSuiteSpec.html) component for the executable you have defined in this guide. See the CUnit support (https://docs.gradle.org/3.5/userguide/native_software.html#native_binaries:cunit) and GoogleTest support (https://docs.gradle.org/3.5/userguide/native_software.html#native_binaries:google_test) sections in the User Guide for more details.
- As there is no 'standard' way of creating documentation for C++ projects, the `cpp` plugin does not offer a task to generate documentation. If you do use the popular Doxygen (http://www.stack.nl/~dimitri/doxygen) tool for documenting C++ code, you may want to have a look at the Doxygen plugin (https://plugins.gradle.org/plugin/org.ysb33r.doxygen) for Gradle

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-executables (https://github.com/gradle-guides/building-cpp-executables/) and we'll get back to you.

Last updated 2017-08-09 20:58:37 UTC