How to set and request specific package version in CMake: Simple Modern CMake Tutorial Part 2

Kohei Otsuka Follow
May 28, 2020 · 4 min read ★

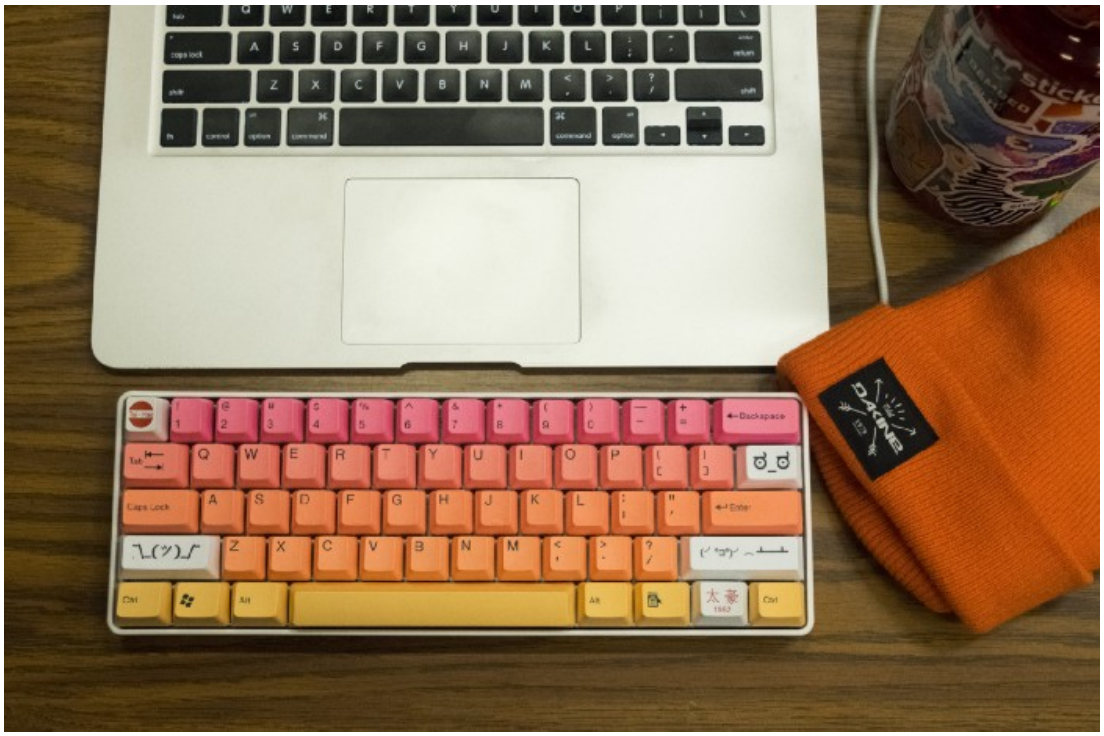

Photo by Paul Esch-Laurent on Unsplash

In my previous post, "*Simple Modern CMake Tutorial Part 1 ~ Short introduction of "Object oriented" Modern CMake*", I explained about the most important aspect of Modern CMake, which is the concept of target and properties. Using Modern CMake features like *target_link_libraries*, *target_include_directories*, we can achieve modular design in CMake packages structure. In the tutorial, we actually created example CMake packages to demonstrate how to export, install a custom CMake package and how to import it from another CMake package (Here is the link to the Github repository).

Now, you know how to export your own package for others to use. After you distribute your package, eventually, other libraries or applications start using your library and depending on it (at least that is how you should expect and prepare). Meanwhile, you are still adding changes to your package (like new features, bug fixes, etc) and updating it. Soon you will notice that you need to do versioning[1] of your package. This is not only for yourself to track software versions for development, but also for users of your package to specify dependency to the correct version of your package. Actually, CMake has facilities to specify the version of package you create and to specify the version of other packages to request.

In this post, I will explain how to set package version when distributing package and how to request a specific version of other package.

Generating a Package Version File

We will use the same code from tutorial part 1 to go through the topic with actual example. Here is the brief overview of the projects (Please read part 1 for more detail).

First, let's look at the CMakeLists.txt file of SomeLibraries (under some_libraries directory). The bold text below is the newly added part in the CMakeLists.txt file.

```
## CMakeLists.txt for SomeLibraries ##

cmake_minimum_required(VERSION 3.5)
project(SomeLibraries VERSION 1.0 LANGUAGES CXX)

add_library(SomeLibraryA src/some_library_a.cpp)

target_include_directories(SomeLibraryA
PUBLIC
$<INSTALL_INTERFACE:include>
$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>)

add_library(SomeLibraryB src/some_library_b.cpp)

target_include_directories(SomeLibraryB
PUBLIC
$<INSTALL_INTERFACE:include>
$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>)

install(TARGETS SomeLibraryA SomeLibraryB
EXPORT SomeLibraries-export
LIBRARY DESTINATION lib
ARCHIVE DESTINATION lib)

install(EXPORT SomeLibraries-export
FILE
SomeLibrariesTargets.cmake
NAMESPACE
SomeLibraries::
DESTINATION
```

```
lib/cmake/SomeLibraries)

install(FILES
${CMAKE_CURRENT_SOURCE_DIR}/include/SomeLibraries/some_library_a.hpp
${CMAKE_CURRENT_SOURCE_DIR}/include/SomeLibraries/some_library_b.hpp
DESTINATION
"include/SomeLibraries")

install(FILES
${CMAKE_CURRENT_SOURCE_DIR}/cmake/SomeLibrariesConfig.cmake
DESTINATION "lib/cmake/SomeLibraries")
```

```
#### The below block was newly added for part 2 ####
#### This is the block specifying & exporting the version of package

include(CMakePackageConfigHelpers)

write_basic_package_version_file(
${CMAKE_CURRENT_SOURCE_DIR}/cmake/SomeLibrariesConfigVersion.cmake
VERSION ${PROJECT_VERSION}
COMPATIBILITY SameMajorVersion)

install(FILES
${CMAKE_CURRENT_SOURCE_DIR}/cmake/SomeLibrariesConfigVersion.cmake
DESTINATION "lib/cmake/SomeLibraries")
```

Let's look at the added texts one by one:

```
include(CMakePackageConfigHelpers)
```

Here it is loading a module called *CMakePackageConfigHelpers*[2]. It is a module included in CMake and it defines utility functions that can be used to generate config files for given packages. We include this module here so that we can use *write_basic_package_version_file* function, which is written in next line.

```
write_basic_package_version_file( ${CMAKE_CURRENT_SOURCE_DIR}/cmake
/SomeLibrariesConfigVersion.cmake VERSION ${PROJECT_VERSION}
COMPATIBILITY SameMajorVersion)
```

write_basic_package_version_file is used to generate a config file that specifies version of the package and the compatibility information(more below). The first argument is name of the file that will be generated. The name has to be *<PackageName>ConfigVersion.cmake*.

VERSION is the package version you like to specify. You can specify like *<major.minor.patch>*. If no VERSION is given, the PROJECT_VERSION variable that was set with *project(SomeLibraries VERSION 1.0 LANGUAGES CXX)* is used (In the above example, I wrote it explicitly for clarity). If this hasn't been set, it gives errors.

COMPATIBILITY is the compatibility information regarding the package. Let's say other package requested the specific version 1.0.0 of your package, but the version installed on the build environment was 2.0.0, then it needs to know if the installed version(2.0.0) is compatible with the requested version(1.0.0) i.e. if the installed version is backward compatible. This is the kind of information only you as the developer of your package know, and you need to provide it. In the case above, *SameMinorVersion* is specified. This

means both major and minor version must be the same as requested, e.g version 2.0 will not be compatible if 2.1 is requested. There are other compatibility types you can specify (*AnyNewerVersion*|*SameMajorVersion*|*SameMinorVersion*|*ExactVersion*). Please refer to the documentation of *CMakePackageConfigHelpers*[2] for more detail.

Lastly, it is copying the generated SomeLibrariesConfigVersion.cmake file to the specified conventional location to be found by *find_package* function later when another package uses it.

```
install(FILES
${CMAKE_CURRENT_SOURCE_DIR}/cmake/SomeLibrariesConfigVersion.cmake
DESTINATION "lib/cmake/SomeLibraries")
```

Request a specific version of a package

As explained before, MyLibraries package uses SomeLibraries package. Let's check its CMakeLists.txt. At this time, only thing that was newly added is the requested package version with *find_package*[3] as you can see below in bold.

```
cmake_minimum_required(VERSION 3.5)
project(MyLibraries VERSION 1.0 LANGUAGES CXX)

find_package(SomeLibraries 1.0 REQUIRED)

add_library(MyLibraryA src/my_library_a.cpp)

target_include_directories(MyLibraryA
PUBLIC
$<INSTALL_INTERFACE:include>
$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>)

target_link_libraries(MyLibraryA
PUBLIC
SomeLibraries::SomeLibraryA
PRIVATE
SomeLibraries::SomeLibraryB)

install(TARGETS MyLibraryA
EXPORT  MyLibraries-export
LIBRARY DESTINATION lib
ARCHIVE DESTINATION lib)

install(EXPORT MyLibraries-export
FILE
MyLibrariesTargets.cmake
NAMESPACE
MyLibraries::
DESTINATION
lib/cmake/MyLibraries)

install(FILES
${CMAKE_CURRENT_SOURCE_DIR}/include/MyLibraries/my_library_a.hpp
```

```
DESTINATION "include/MyLibraries")

install(FILES
${CMAKE_CURRENT_SOURCE_DIR}/cmake/MyLibrariesConfig.cmake
DESTINATION "lib/cmake/MyLibraries")
```

With *find_package(SomeLibraries 1.0 REQUIRED)*, it is requesting *SomeLibraries* package with specific version 1.0. Then it finds the SomeLibrariesConfigVersion.cmake file we generated for *SomeLibraries* and checks the version and the compatibility information. If it is not matching to what is requested, it gives error.

Summary

In this post, I explained how to set package version when distributing package and how to request a specific version of other package with CMake.

[1]: Software versioning, https://en.wikipedia.org/wiki/Software_versioning

[2]: CMakePackageConfigHelpers, https://cmake.org/cmake/help/latest/module/CMakePackageConfigHelpers.html

[3]: *find_package*(). https://cmake.org/cmake/help/v3.0/command/find_package.html