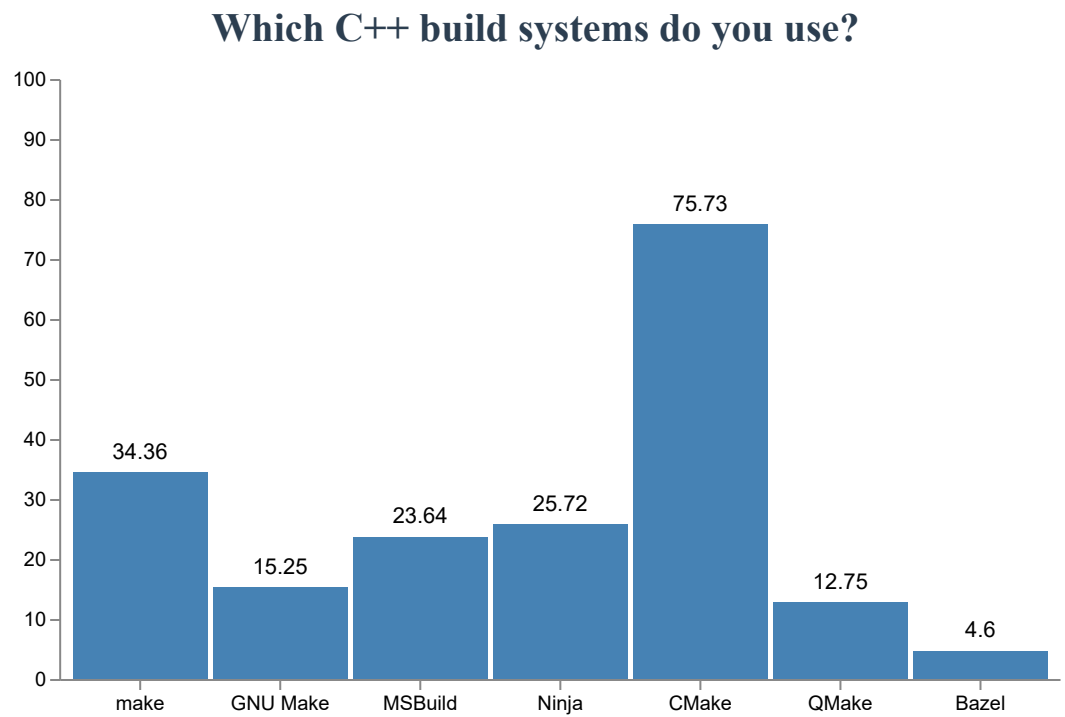
Clean CMake for C++ (library) developers

Use CMake

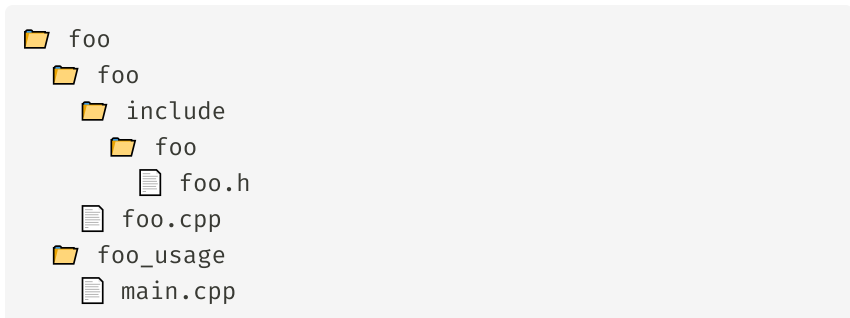
Meeting C++ developer survey



Understand how CMake works internally

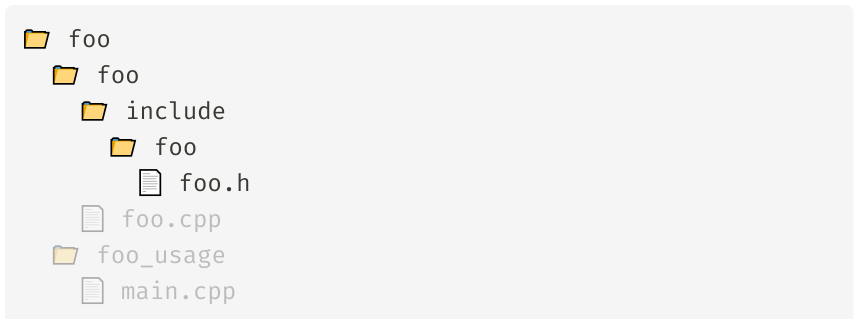
Default library setup

`Source directory`



Default library setup

`Source directory`

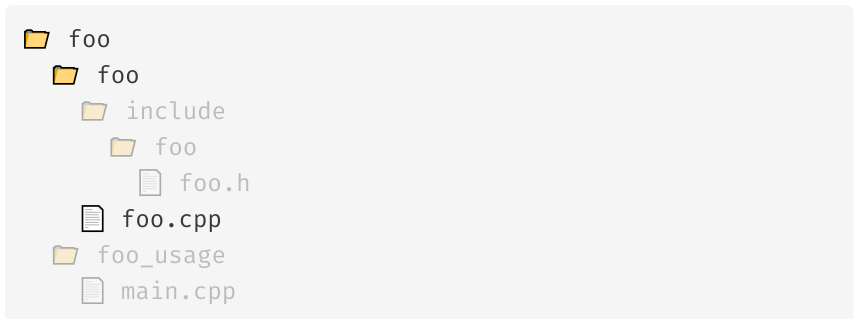


`foo.h`

```
1  #pragma once
2
3  namespace foo
4  {
5    int foo();
6  }
```

Default library setup

`Source directory`

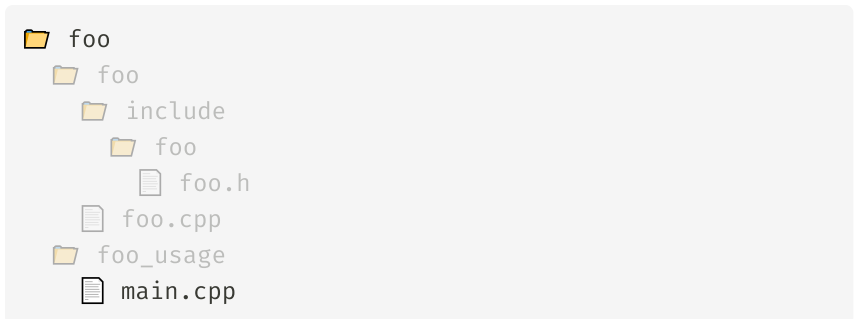


`foo.cpp`

```cpp
#include <foo/foo.h>

namespace foo
{
  int foo()
  {
    return 42;
  }
}
```

Default library setup

`Source directory`

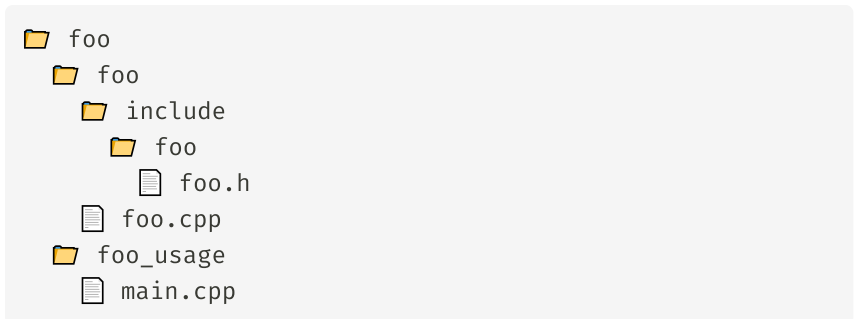


`main.cpp`

```cpp
#include <iostream>
#include <foo/foo.h>

int main()
{
  std::cout << "foo: " << foo::foo() << std::endl;
}
```

How to compile `foo_usage(.exe)` ?

`Source directory`



Create a shared library

```
g++
  ./foo/foo.cpp
  -o libfoo.so
  -fPIC -shared
  -I./foo/include
```

Create the executable

```
g++
  ./foo_usage/main.cpp
  -o foo_usage
  -I./foo/include
  -L. -lfoo
```

Old school CMake files

`CMakeLists.txt`

```
# Specify the minimum required version of CMake
cmake_minimum_required(VERSION 2.8)

# Name the project
project(foo)

set(CMAKE_CXX_FLAGS "-fpic")

# Set include directories, if needed
include_directories(${CMAKE_SOURCE_DIR}/foo/include)

# Specify the executable target and source files
add_library(foo SHARED foo/foo.cpp)
```

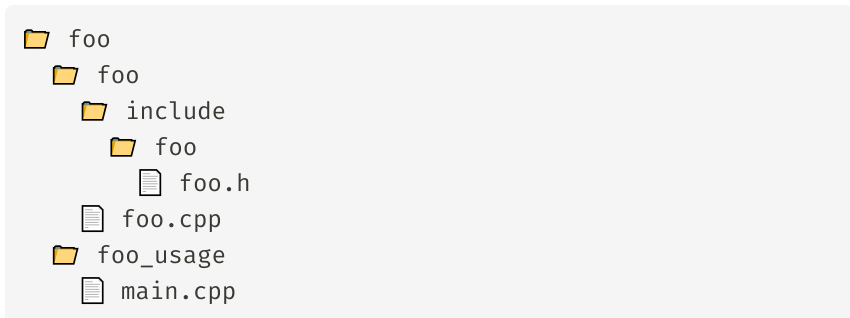
Compiler Command

```
g++
  ./foo/foo.cpp
  -o libfoo.so
  -fPIC
  -shared
  -I./foo/include
```

!!! DO NOT WRITE CMAKE FILES LIKE THIS !!!

Creating a library using Modern CMake

`Source directory`



Creating a library using Modern CMake

`Source directory`

Creating a library using Modern CMake

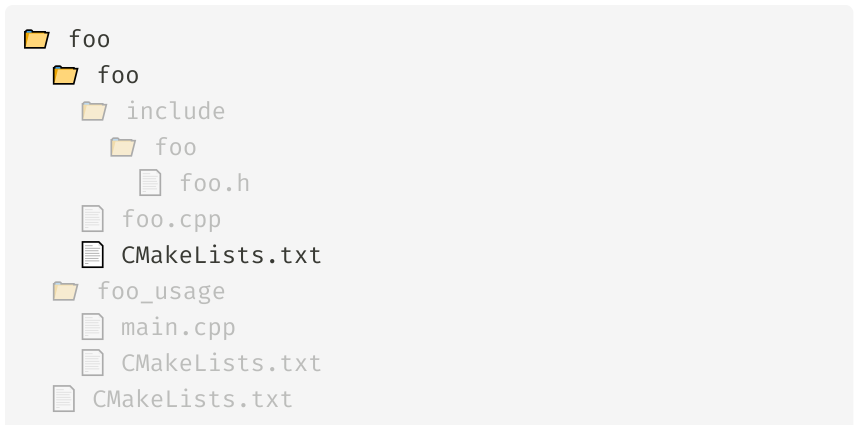
`Source directory`

`CMakeLists.txt`

```cmake
cmake_minimum_required(VERSION 3.25)

project(foo)

add_subdirectory(foo)
add_subdirectory(foo_usage)
```

Creating a library using Modern CMake

`Source directory`



`CMakeLists.txt`

```cmake
add_library(foo)

target_sources(foo
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)
```

Creating a library using Modern CMake

`Source directory`

`CMakeLists.txt`

```
1  add_executable(foo_usage)
2
3  target_sources(foo_usage
4    PRIVATE
5      main.cpp
6  )
7
8  target_link_libraries(foo_usage
9    PRIVATE
10     foo
11 )
```

CMake targets and their properties

`CMakeLists.txt`

```
1   add_library(foo)
2
3   target_sources(foo
4     PUBLIC
5       FILE_SET foo_headers
6       TYPE HEADERS
7       BASE_DIRS ./include
8       FILES include/foo/foo.h
9     PRIVATE
10      foo.cpp
11  )
```

`Targets & Properties`

```
foo
  SOURCES
    include/foo/foo.h
    foo.cpp
  INTERFACE_SOURCES
    include/foo/foo.h
  INCLUDE_DIRECTORIES
    ./include
  INTERFACE_INCLUDE_DIRECTORIES
    ./include
  LINK_LIBRARIES
  INTERFACE_LINK_LIBRARIES
```

CMake targets and their properties

`CMakeLists.txt`

```
1   add_executable(foo_usage)
2
3   target_sources(foo_usage
4     PRIVATE
5       main.cpp
6   )
7
8   target_link_libraries(foo_usage
9     PRIVATE
10      foo
11  )
```

`foo Properties`

```
foo
  SOURCES
    include/foo/foo.h
    foo.cpp
  INTERFACE_SOURCES
    include/foo/foo.h
  INCLUDE_DIRECTORIES
    ./include
  INTERFACE_INCLUDE_DIRECTORIES
    ./include
  LINK_LIBRARIES
  INTERFACE_LINK_LIBRARIES
```

`foo_usage Properties`

```
foo_usage
  SOURCES
    main.cpp
    <foo>/include/foo/foo.h
  INTERFACE_SOURCES
  INCLUDE_DIRECTORIES
    <foo>/include
  INTERFACE_INCLUDE_DIRECTORIES
  LINK_LIBRARIES
    <foo>/foo.lib
  INTERFACE_LINK_LIBRARIES
```

Package your library with CMake

Package your library with CMake

- There is no such thing a a C++ package
- We need to collect
  - The compiled library & header files
- We need to additionaly provide
  - Information on how to consume the library
- The "package" should be relocatable

Package your library with CMake

`Source directory`

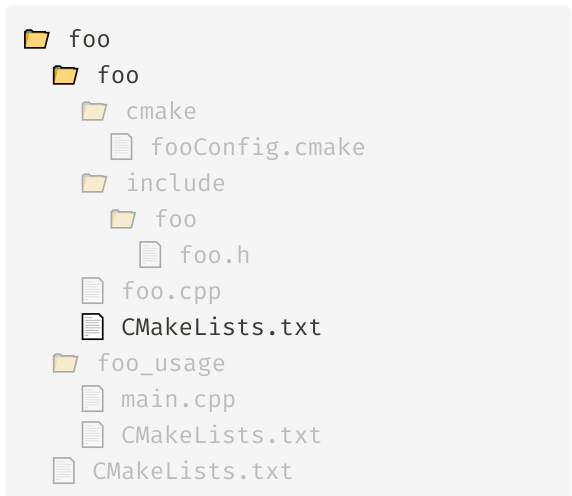
```
foo
  foo
    cmake
      fooConfig.cmake
    include
      foo
        foo.h
    foo.cpp
    CMakeLists.txt
  foo_usage
    main.cpp
    CMakeLists.txt
  CMakeLists.txt
```

`Install directory`

```
_install
```

Package your library with CMake

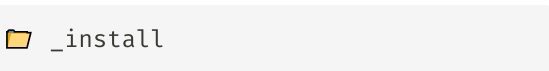
`Source directory`



`CMakeLists.txt`

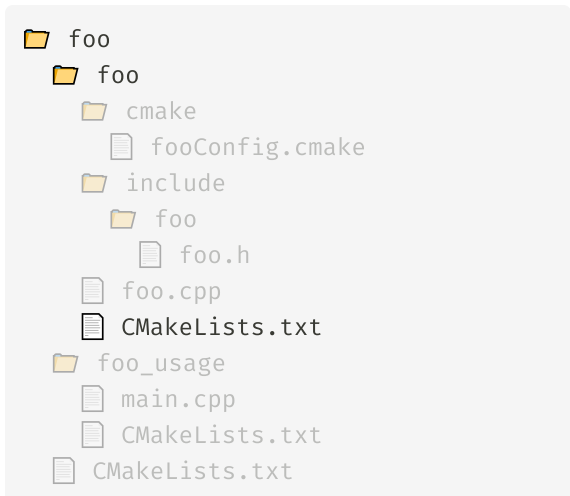
```
add_library(foo)

target_sources(foo
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)
```

`Install directory`



Package your library with CMake

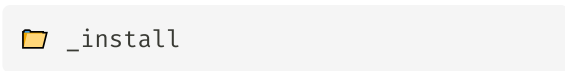
`Source directory`



`CMakeLists.txt`

```cmake
add_library(foo)

target_sources(foo
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)

install(
  TARGETS foo
  EXPORT fooTargets
  FILE_SET foo_headers
)
```

`Install directory`



Package your library with CMake

`Source directory`

```
foo
  foo
    cmake
      fooConfig.cmake
    include
      foo
        foo.h
    foo.cpp
    CMakeLists.txt
  foo_usage
    main.cpp
    CMakeLists.txt
  CMakeLists.txt
```

`CMakeLists.txt`

```
add_library(foo)

target_sources(foo
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)

install(
  TARGETS foo
  EXPORT fooTargets
  FILE_SET foo_headers
)
```

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
```

Package your library with CMake

`Source directory`

```
foo
  foo
    cmake
      fooConfig.cmake
    include
      foo
        foo.h
    foo.cpp
    CMakeLists.txt
  foo_usage
    main.cpp
    CMakeLists.txt
  CMakeLists.txt
```

`CMakeLists.txt`

```
add_library(foo)

...

install(
  TARGETS foo
  EXPORT fooTargets
  FILE_SET foo_headers
)

install(
  EXPORT fooTargets
  DESTINATION share/cmake/foo
  NAMESPACE foo::
)
```

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
```

Package your library with CMake

`Source directory`

```
foo
  foo
    cmake
      fooConfig.cmake
    include
      foo
        foo.h
    foo.cpp
    CMakeLists.txt
  foo_usage
    main.cpp
    CMakeLists.txt
  CMakeLists.txt
```

`CMakeLists.txt`

```
add_library(foo)

...

install(
  TARGETS foo
  EXPORT fooTargets
  FILE_SET foo_headers
)

install(
  EXPORT fooTargets
  DESTINATION share/cmake/foo
  NAMESPACE foo::
)
```

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
  share
    cmake
      foo
        fooTargets.cmake
        fooTargets-Release.cmake
```

Package your library with CMake

`Source directory`

```
foo
  foo
    cmake
      fooConfig.cmake
    include
      foo
        foo.h
    foo.cpp
    CMakeLists.txt
  foo_usage
    main.cpp
    CMakeLists.txt
  CMakeLists.txt
```

`CMakeLists.txt`

```
add_library(foo)

...

install(
  FILES
    ./cmake/fooConfig.cmake
  DESTINATION
    share/cmake/foo
)
```

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
  share
    cmake
      foo
        fooTargets.cmake
        fooTargets-Release.cmake
```

Package your library with CMake

`Source directory`

```
foo
  foo
    cmake
      fooConfig.cmake
    include
      foo
        foo.h
    foo.cpp
    CMakeLists.txt
  foo_usage
    main.cpp
    CMakeLists.txt
  CMakeLists.txt
```

`CMakeLists.txt`

```
add_library(foo)

...

install(
  FILES
    ./cmake/fooConfig.cmake
  DESTINATION
    share/cmake/foo
)
```

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
  share
    cmake
      foo
        fooTargets.cmake
        fooTargets-Release.cmake
        fooConfig.cmake
```

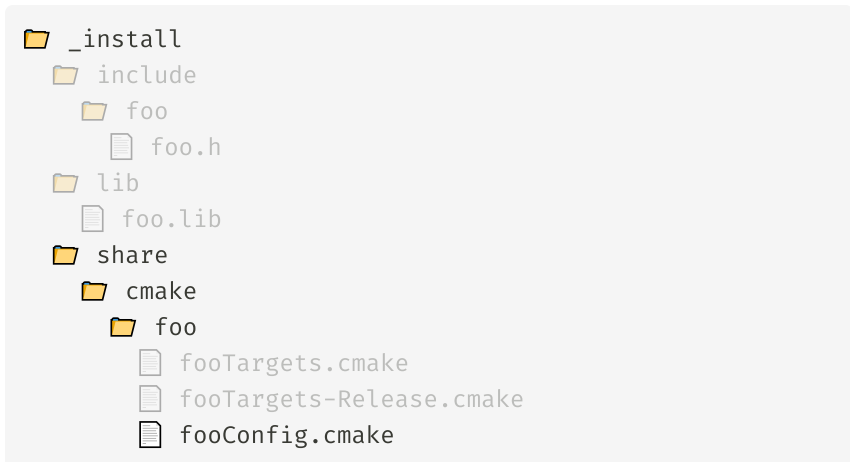
Package your library with CMake

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
  share
    cmake
      foo
        fooTargets.cmake
        fooTargets-Release.cmake
        fooConfig.cmake
```

Package your library with CMake

`Install directory`



`FooConfig.cmake`

```cmake
include(${CMAKE_CURRENT_LIST_DIR}/fooTargets.cmake)
```

Package your library with CMake

`Install directory`

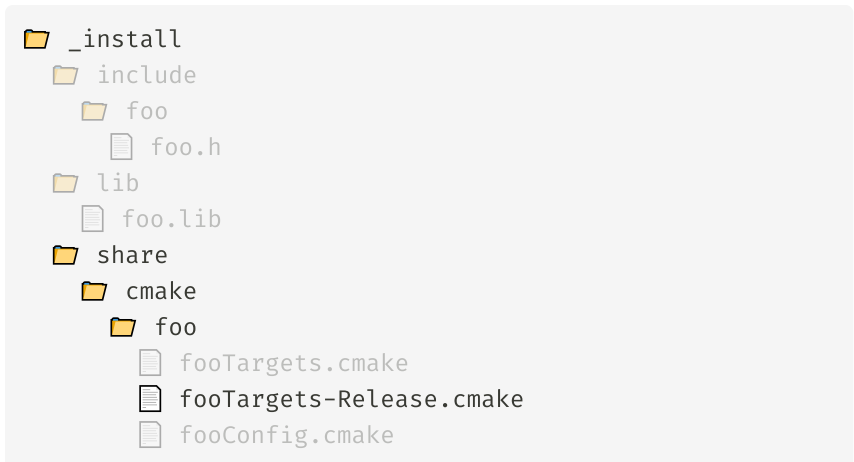
```
_install
  include
    foo
      foo.h
  lib
    foo.lib
  share
    cmake
      foo
        fooTargets.cmake
        fooTargets-Release.cmake
        fooConfig.cmake
```

`FooTargets.cmake`

```
...

add_library(foo::foo STATIC IMPORTED)

if(NOT CMAKE_VERSION VERSION_LESS "3.23.0")
  target_sources(foo::foo
    INTERFACE
      FILE_SET "foo_headers"
      TYPE "HEADERS"
      BASE_DIRS "${_IMPORT_PREFIX}/include"
      FILES "${_IMPORT_PREFIX}/include/foo/foo.h"
  )
else()
  set_property(TARGET foo::foo
    APPEND PROPERTY INTERFACE_INCLUDE_DIRECTORIES
      "${_IMPORT_PREFIX}/include"
  )
endif()

...
```

Package your library with CMake

`Install directory`



`FooTargets-Release.cmake`

```
...

# Import target "foo::foo" for configuration "Release"
set_property(TARGET foo::foo
  APPEND PROPERTY IMPORTED_CONFIGURATIONS RELEASE)
set_target_properties(foo::foo PROPERTIES
  IMPORTED_LINK_INTERFACE_LANGUAGES_RELEASE
    "CXX"
  IMPORTED_LOCATION_RELEASE
    "${_IMPORT_PREFIX}/lib/foo.lib"
  )

...
```

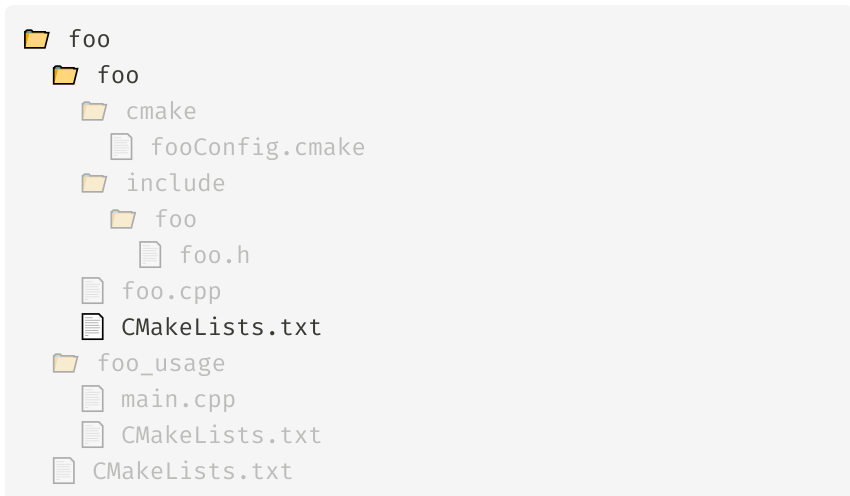
Package your library with CMake

`Install directory`

```
_install
  include
    foo
      foo.h
  lib
    foo.lib
  share
    cmake
      foo
        fooTargets.cmake
        fooTargets-Release.cmake
        fooConfig.cmake
```

Package your library with CMake

`Source directory`

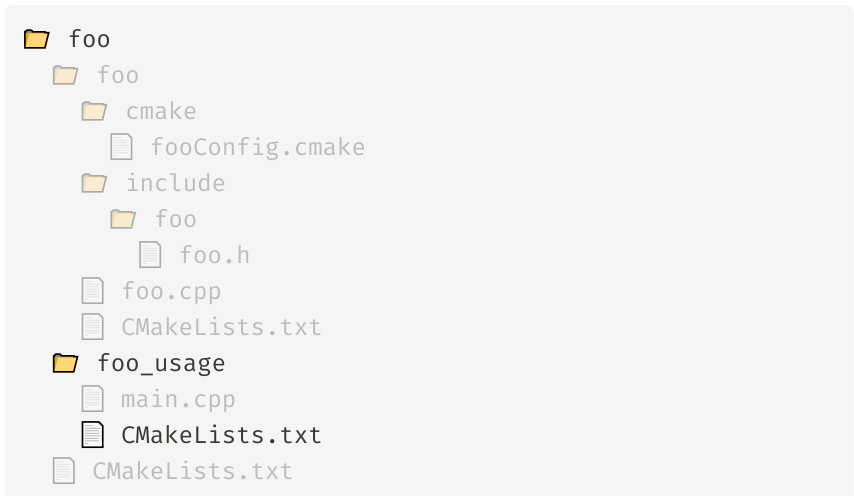


`CMakeLists.txt`

```
1  add_library(foo)
2  add_library(foo::foo ALIAS foo)
3
4  ...
```

Package your library with CMake

`Source directory`



`CMakeLists.txt`

```cmake
find_package(foo REQUIRED)

add_executable(foo_usage)

target_sources(foo_usage
  PRIVATE
    main.cpp
)

target_link_libraries(foo_usage
  PRIVATE
    foo::foo
)
```

Package your library with CMake

- Use CMake to package your library.
- Document how to build the library
- Document (in code) how to consume the library

Handle C++ dependencies the right way

Handle C++ dependencies the right way

- Don’t vendor dependencies

Don’t vendor dependencies

Not vendoring dependencies

`CMakeLists.txt`

```
1  cmake_minimum_required(VERSION 3.25)
2  project(foo)
3
4  find_package(GTest REQUIRED)
5  find_package(Protobuf REQUIRED)
6
7  add_subdirectory(foo)
8  add_subdirectory(foo_usage)
```

Vendoring dependencies

`CMakeLists.txt`

```
1  cmake_minimum_required(VERSION 3.25)
2  project(foo)
3
4  add_subdirectory(foo)
5  add_subdirectory(foo_usage)
6
7  add_subdirectory(thirdparty/gtest)
8
9  FetchContent_Declare(Protobuf
10   GIT_REPOSITORY  https://github.com/ … /protobuf
11   GIT_TAG         v3.26.1
12 )
13 FetchContent_MakeAvailable(Protobuf)
```

Handle C++ dependencies the right way

- Don’t vendor dependencies.
  - Use `dependency_providers` (>= CMake 3.24) instead
  - Pre 3.24, write custom `Find<dependency>.cmake` files

Handle C++ dependencies the right way

- Don’t vendor dependencies
  - Use `dependency_providers` (>= CMake 3.24) instead
  - Pre 3.24, write custom `Find<dependency>.cmake` files
- Handle dependencies the "modern" way

Handle dependencies the "modern" way

Modern CMake dependency usage

`CMakeLists.txt`

```
find_package(Protobuf REQUIRED)

add_library(foo ... )

target_link_libraries(foo
  PUBLIC
    Protobuf::libprotobuf
)
```

Old style dependency usage

`CMakeLists.txt`

```
find_package(Protobuf REQUIRED)

add_library(foo ... )

target_include_directories(foo
  PUBLIC
    ${Protobuf_INCLUDE_DIRS}
)

target_link_libraries(foo
  PUBLIC
    ${Protobuf_LIBRARIES}
)
```

Handle C++ dependencies the right way

- Don’t vendor dependencies
  - Use `dependency_providers` (>= CMake 3.24) instead
  - Pre 3.24, write custom `Find<dependency>.cmake` files
- Handle dependencies the "modern" way
- Be explicit when handling dependencies

Be explicit when handling dependencies

Implicit dependency usage

`CMakeLists.txt`

```
find_package(JPEG)

if (JPEG_FOUND)
  add_library(foo_jpeg)
  # ...
  target_link_libraries(foo_jpeg
    PRIVATE
      jpeg::jpeg
  )
endif()
```

Explicit dependency usage

`CMakeLists.txt`

```
option(FOO_HAS_JPEG "Build foo with JPEG" ON)

if (FOO_HAS_JPEG)
  find_package(JPEG REQUIRED)
  add_library(foo_jpeg)
  # ...
  target_link_libraries(foo_jpeg
    PRIVATE
      jpeg::jpeg
  )
endif()
```

Handle C++ dependencies the right way

- Don’t vendor dependencies
  - Use `dependency_providers` (>= CMake 3.24) instead
  - Pre 3.24, write custom `Find<dependency>.cmake` files
- Handle dependencies the "modern" way
- Be explicit when handling dependencies

Keep your CMakeLists.txt files clean

Keep your CMakeLists.txt files clean

- Treat CMake code like production code

Keep your CMakeLists.txt files clean

- Treat CMake code like production code
- **Distinguish build settings from usage requirements**

Target Property initialization

`CMakeLists.txt`

```
1  set(CMAKE_CXX_STANDARD 17)
2  add_compile_options(-Wall -Wextra)
3
4  add_library(foo)
```

`Targets & Properties`

```
foo
  CXX_STANDARD
    17
  COMPILE_OPTIONS
    -Wall -Wextra
```

What is the difference between this code?

`CMakeLists.txt`

```
set(CMAKE_CXX_STANDARD 17)

add_library(foo)

target_sources(foos
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)
```

`CMakeLists.txt`

```
add_library(foo)

target_sources(foo
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)

target_compile_features(foo PUBLIC cxx_std_17)
```

What is the difference between this code?

`CMakeLists.txt`

```cmake
set(CMAKE_CXX_STANDARD 17)

add_library(foo)

target_sources(foos
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)
```

```cmake
# Create imported target foo::foo
add_library(foo::foo STATIC IMPORTED)
```

`CMakeLists.txt`

```cmake
add_library(foo)

target_sources(foo
  PUBLIC
    FILE_SET foo_headers
    TYPE HEADERS
    BASE_DIRS ./include
    FILES include/foo/foo.h
  PRIVATE
    foo.cpp
)

target_compile_features(foo PUBLIC cxx_std_17)
```

```cmake
# Create imported target foo::foo
add_library(foo::foo STATIC IMPORTED)

set_target_properties(foo::foo PROPERTIES
  INTERFACE_COMPILE_FEATURES "cxx_std_17"
)
```

Which code is better?

`main.cpp`

```
#include "C:/Program Files/ ... /iostream"

main()
{
  std::cout << "Hello" << std::endl;
}
```

`main.cpp`

```
#include <iostream>

main()
{
  std::cout << "Hello" << std::endl;
}
```

Keep your CMakeLists.txt files clean

- Treat CMake code like production code
- Distinguish build settings from usage requirements
- **Inject build settings from the outside**

How that applies to CMake

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.25)

project(foo)

if(UNIX)
  message(STATUS "GCC detected - Adding flags")
  add_compile_options(-Wall -Wextra)
endif()

add_subdirectory(foo)
add_subdirectory(foo_usage)
```

`CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.25)

project(foo)

add_subdirectory(foo)
add_subdirectory(foo_usage)
```

`CMakePresets.json`

```
{
  "version": 3,
  "configurePresets": [
    {
      "name": “unix",
      "displayName": "Default",
      "cacheVariables": {
        "COMPILE_OPTIONS": “-Wall -Wextra"
      }
}]}
```

Keep your CMakeLists.txt files clean

- Treat CMake code like production code
- Distinguish build settings from usage requirements
- Inject build settings from the outside
- **Think carefully about what are usage requirements, and what are not**

The 5 ultimate Steps to clean CMake code

The 5 ultimate Steps to clean CMake code

- Use CMake
- Understand how CMake works internally
- Package your library with CMake
- Handle C++ dependencies the right way
- Keep your CMakeLists.txt files clean

Thank you for your attention