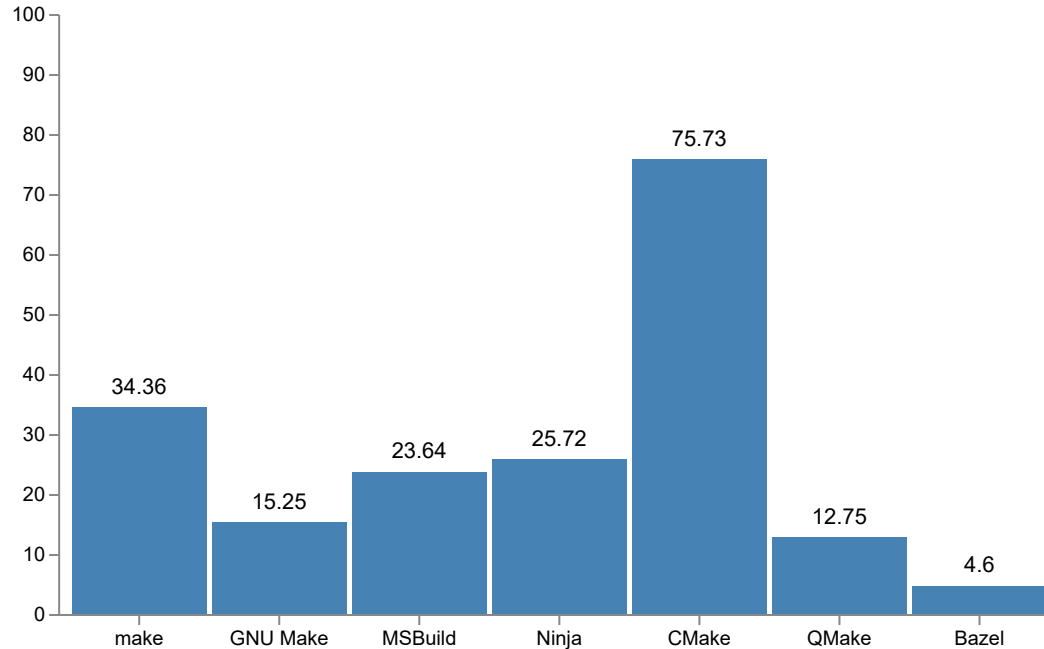


Clean CMake for C++ (library) developers

Use CMake

Meeting C++ developer survey

Which C++ build systems do you use?



Understand how CMake works internally

Default library setup

Source directory



```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   └── foo.cpp
└── foo_usage
    └── main.cpp
```

The diagram illustrates a directory tree for a C++ library. The root directory is 'foo'. Inside 'foo', there is a sub-directory 'foo'. Within this 'foo' sub-directory, there is an 'include' directory containing a 'foo' sub-directory with a 'foo.h' header file. Also in the 'foo' sub-directory are 'foo.cpp' and 'foo_usage' sub-directory. The 'foo_usage' sub-directory contains a 'main.cpp' file.

Default library setup

Source directory



```
foo
├── foo
│   └── include
│       └── foo
│           └── foo.h
├── foo.cpp
├── foo_usage
└── main.cpp
```

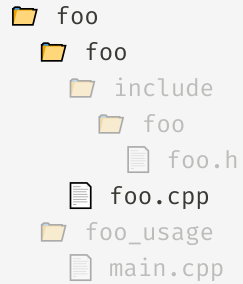
A directory tree diagram showing the source directory structure. The root directory is 'foo'. Inside 'foo', there is a subdirectory 'foo'. Inside 'foo/foo', there is a subdirectory 'include'. Inside 'foo/foo/include', there is a subdirectory 'foo'. Inside 'foo/foo/include/foo', there is a file 'foo.h'. At the root 'foo' level, there are also files 'foo.cpp' and 'main.cpp', and a subdirectory 'foo_usage'.

foo.h

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

Default library setup

Source directory



```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   └── foo.cpp
├── foo_usage
└── main.cpp
```

A directory tree diagram showing the source directory structure. The root directory is 'foo'. Inside 'foo', there is a sub-directory 'foo'. Inside 'foo/foo', there is a sub-directory 'include' which contains a sub-directory 'foo' containing the file 'foo.h'. Also inside 'foo/foo' is the file 'foo.cpp'. At the root level, there is also a sub-directory 'foo_usage' and a file 'main.cpp'.


foo.cpp

```
1  #include <foo/foo.h>
2
3  namespace foo
4  {
5      int foo()
6      {
7          return 42;
8      }
9  }
```

Code listing for foo.cpp showing the inclusion of the header file, namespace declaration, and the implementation of the foo() function.

Default library setup

Source directory



```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   └── foo.cpp
├── foo_usage
└── main.cpp
```

A directory tree diagram showing the source directory structure. The root is a folder named 'foo'. Inside 'foo', there is another folder 'foo'. Inside this 'foo' folder, there is a sub-folder 'include' which contains a sub-folder 'foo' with a file 'foo.h'. Also inside the 'foo' folder, there is a file 'foo.cpp'. Below the 'foo' folder, there is a folder 'foo_usage' and a file 'main.cpp'.

main.cpp

```
1  #include <iostream>
2  #include <foo/foo.h>
3
4  int main()
5  {
6      std::cout << "foo: " << foo::foo() << std::endl;
7  }
```


How to compile `foo_usage(.exe)` ?

Source directory

```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   └── foo.cpp
├── foo_usage
└── main.cpp
```

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

```
1  # Specify the minimum required version of CMake
2  cmake_minimum_required(VERSION 2.8)
3
4  # Name the project
5  project(foo)
6
7  set(CMAKE_CXX_FLAGS "-fpic")
8
9  # Set include directories, if needed
10 include_directories(${CMAKE_SOURCE_DIR}/foo/include)
11
12 # Specify the executable target and source files
13 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



```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   └── foo.cpp
└── foo_usage
    └── main.cpp
```

The diagram illustrates the source directory structure for a C++ library. It shows a root directory named 'foo'. Inside 'foo', there is a sub-directory 'foo'. Within this 'foo' sub-directory, there is an 'include' directory containing a 'foo' sub-directory with a 'foo.h' header file. Additionally, there is a 'foo.cpp' source file in the 'foo' sub-directory. Outside the 'foo' sub-directory, there is a 'foo_usage' directory containing a 'main.cpp' source file.

Creating a library using Modern CMake

Source directory

```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   ├── foo.cpp
│   └── CMakeLists.txt
├── foo_usage
│   ├── main.cpp
│   └── CMakeLists.txt
└── CMakeLists.txt
```

Creating a library using Modern CMake

Source directory

```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   ├── foo.cpp
│   └── CMakeLists.txt
├── foo_usage
│   ├── main.cpp
│   └── CMakeLists.txt
└── CMakeLists.txt
```

CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.25)
2
3  project(foo)
4
5  add_subdirectory(foo)
6  add_subdirectory(foo_usage)
```

Creating a library using Modern CMake

Source directory

```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   ├── foo.cpp
│   └── CMakeLists.txt
├── foo_usage
│   ├── main.cpp
│   └── CMakeLists.txt
└── CMakeLists.txt
```

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    )
```

Creating a library using Modern CMake

Source directory

```
foo
├── foo
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   ├── foo.cpp
│   └── CMakeLists.txt
├── foo_usage
│   ├── main.cpp
│   └── CMakeLists.txt
└── CMakeLists.txt
```

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 as a C++ package
- We need to collect
 - The compiled library & header files
- We need to additionally 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

```
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 directory

```
_install
```

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
```

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

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

FooConfig.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

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

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

```
foo
├── foo
│   ├── cmake
│   │   └── fooConfig.cmake
│   ├── include
│   │   └── foo
│   │       └── foo.h
│   ├── foo.cpp
│   └── CMakeLists.txt
├── foo_usage
│   ├── main.cpp
│   └── CMakeLists.txt
└── CMakeLists.txt
```

CMakeLists.txt

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

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

```
1  find_package(foo REQUIRED)
2
3  add_executable(foo_usage)
4
5  target_sources(foo_usage
6     PRIVATE
7     main.cpp
8 )
9
10 target_link_libraries(foo_usage
11     PRIVATE
12     foo::foo
13 )
```

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` (\geq 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

```
1 find_package(Protobuf REQUIRED)
2
3 add_library(foo ... )
4
5 target_link_libraries(foo
6     PUBLIC
7     Protobuf::libprotobuf
8 )
```

Old style dependency usage

CMakeLists.txt

```
1 find_package(Protobuf REQUIRED)
2
3 add_library(foo ... )
4
5 target_include_directories(foo
6     PUBLIC
7     ${Protobuf_INCLUDE_DIRS}
8 )
9
10 target_link_libraries(foo
11     PUBLIC
12     ${Protobuf_LIBRARIES}
13 )
```

Handle C++ dependencies the right way

- Don't vendor dependencies
 - Use `dependency_providers` (\geq 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

```
1 find_package(JPEG)
2
3 if (JPEG_FOUND)
4     add_library(foo_jpeg)
5     # ...
6     target_link_libraries(foo_jpeg
7         PRIVATE
8         jpeg::jpeg
9     )
10 endif()
```

Explicit dependency usage

CMakeLists.txt

```
1 option(FOO_HAS_JPEG "Build foo with JPEG" ON)
2
3 if (FOO_HAS_JPEG)
4     find_package(JPEG REQUIRED)
5     add_library(foo_jpeg)
6     # ...
7     target_link_libraries(foo_jpeg
8         PRIVATE
9         jpeg::jpeg
10    )
11 endif()
```

Handle C++ dependencies the right way

- Don't vendor dependencies
 - Use `dependency_providers` (\geq 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

```
1  set(CMAKE_CXX_STANDARD 17)
2
3  add_library(foo)
4
5  target_sources(foo
6     PUBLIC
7     FILE_SET foo_headers
8     TYPE HEADERS
9     BASE_DIRS ./include
10    FILES include/foo/foo.h
11    PRIVATE
12    foo.cpp
13 )
```

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 )
12
13 target_compile_features(foo PUBLIC cxx_std_17)
```

What is the difference between this code?

CMakeLists.txt

```
1  set(CMAKE_CXX_STANDARD 17)
2
3  add_library(foo)
4
5  target_sources(foo
6    PUBLIC
7      FILE_SET foo_headers
8      TYPE HEADERS
9      BASE_DIRS ./include
10     FILES include/foo/foo.h
11     PRIVATE
12     foo.cpp
13 )
```

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

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 )
12
13 target_compile_features(foo PUBLIC cxx_std_17)
```

```
1  # Create imported target foo::foo
2  add_library(foo::foo STATIC IMPORTED)
3
4  set_target_properties(foo::foo PROPERTIES
5    INTERFACE_COMPILE_FEATURES "cxx_std_17"
6  )
```

Which code is better?

main.cpp

```
1  #include "C:/Program Files/.../iostream"
2
3  main()
4  {
5      std::cout << "Hello" << std::endl;
6  }
```

main.cpp

```
1  #include <iostream>
2
3  main()
4  {
5      std::cout << "Hello" << std::endl;
6  }
```

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

```
1  cmake_minimum_required(VERSION 3.25)
2
3  project(foo)
4
5  if(UNIX)
6      message(STATUS "GCC detected - Adding flags")
7      add_compile_options(-Wall -Wextra)
8  endif()
9
10 add_subdirectory(foo)
11 add_subdirectory(foo_usage)
```

CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.25)
2
3  project(foo)
4
5  add_subdirectory(foo)
6  add_subdirectory(foo_usage)
```

CMakePresets.json

```
1  {
2      "version": 3,
3      "configurePresets": [
4          {
5              "name": "unix",
6              "displayName": "Default",
7              "cacheVariables": {
8                  "COMPILE_OPTIONS": "-Wall -Wextra"
9              }
10         }
11     ]
12 }
```

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