

OH NO! MORE Modern CMake



Deniz Bahadir  BENOCS

✉ cmake@deniz.bahadir.email

 @DenizThatMenace

 <https://github.com/Bagira80/More-Modern-CMake/>



OH NO! MORE Modern CMake



Deniz Bahadir  BENOCS

✉ cmake@deniz.bahadir.email

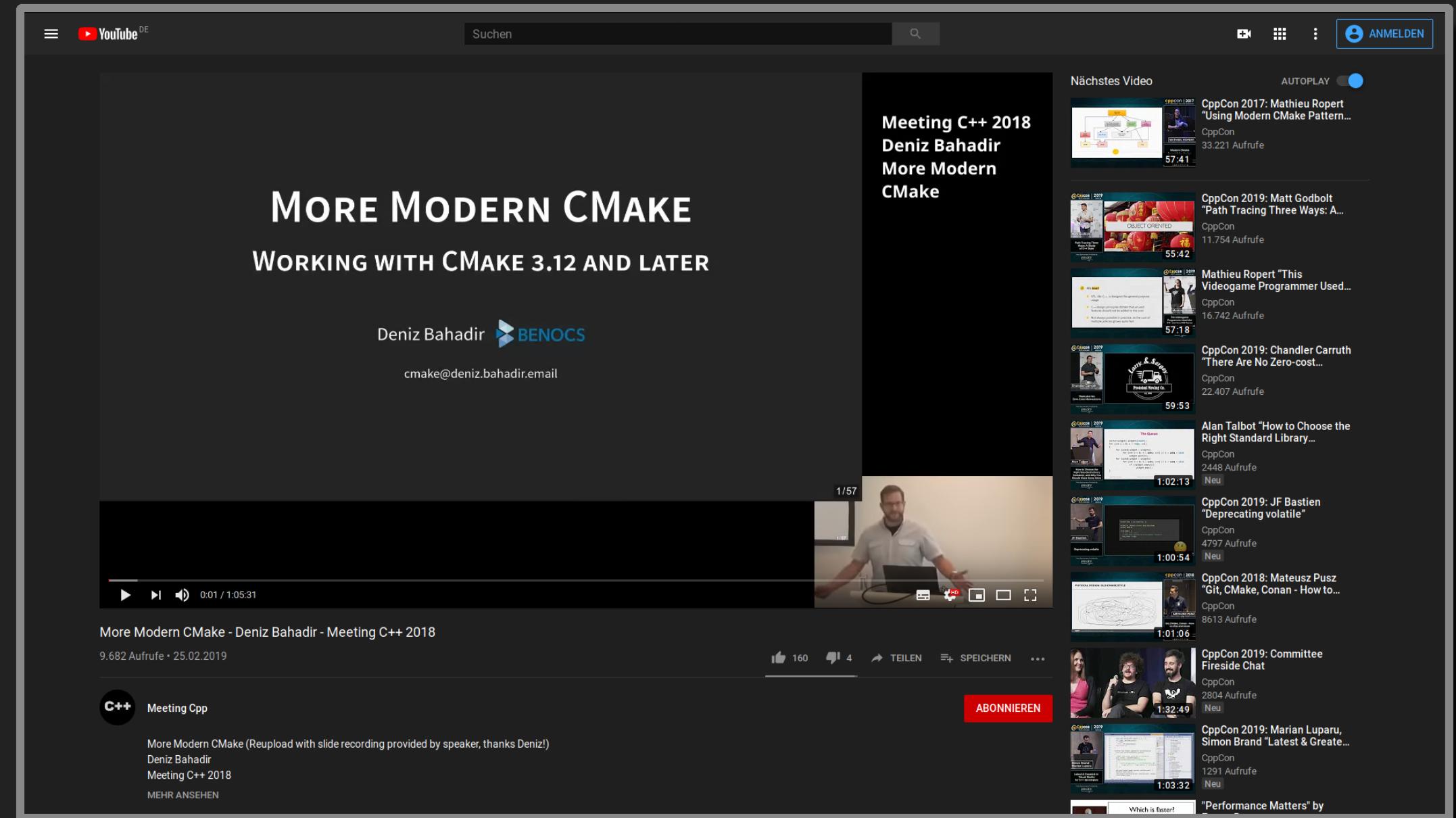
 @DenizThatMenace

 <https://github.com/Bagira80/More-Modern-CMake/>



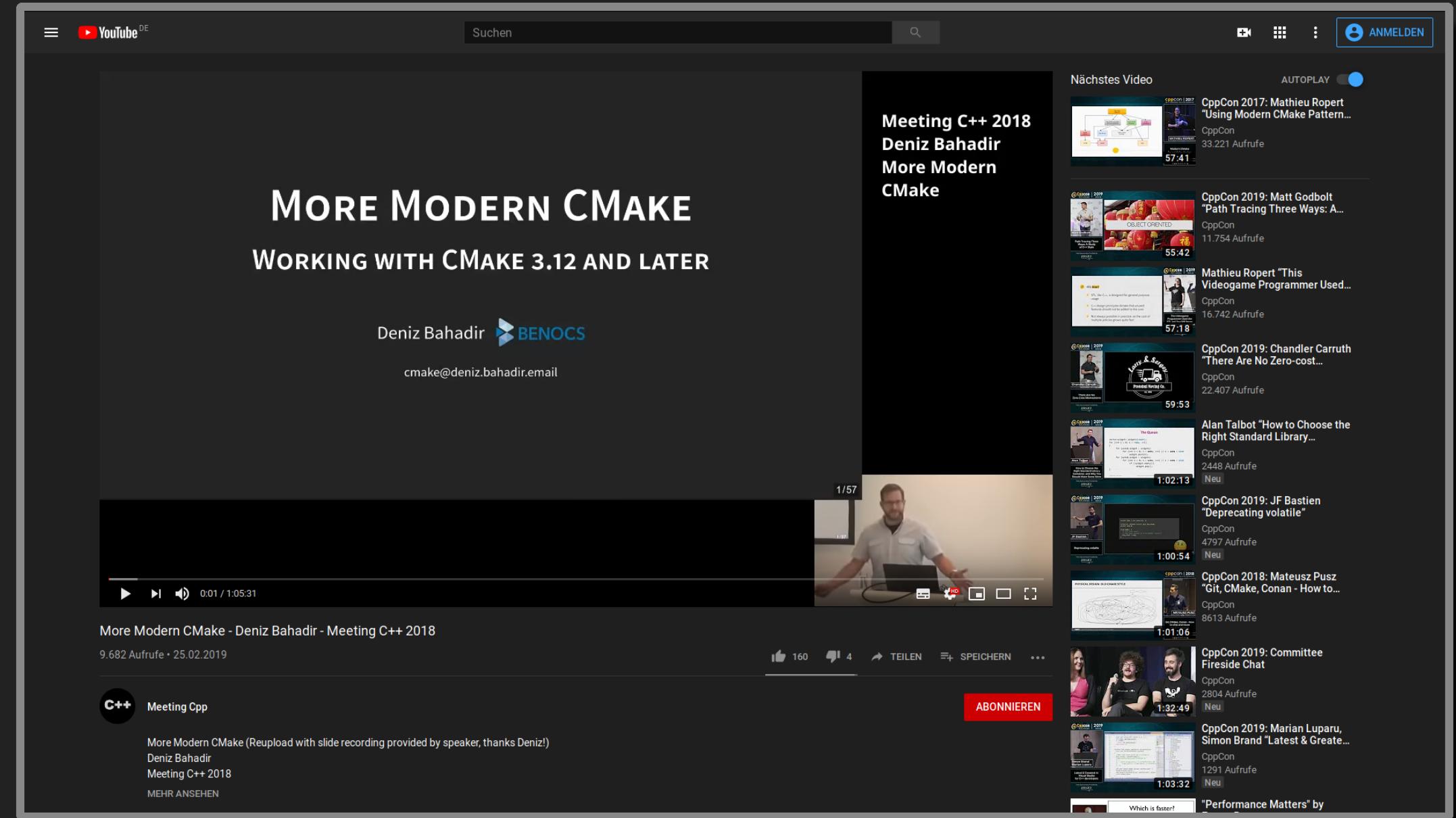
source: <http://gph.is/1fvlBsY>

A SEQUEL TO LAST YEAR'S TALK



<https://youtu.be/y7ndUhdQuU8>

A SEQUEL TO LAST YEAR'S TALK



<https://youtu.be/y7ndUhdQuU8>

- Who has seen last year's talk?

A SHORT RECAP

WHAT IS (MODERN) CMAKE?

- CMake
- Modern CMake

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
- Modern CMake

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
 - supports generating build-system input files for multiple languages.
 - **C/C++**, FORTRAN, C#, CUDA...
- Modern CMake

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
 - supports generating build-system input files for multiple languages.
 - **C/C++**, FORTRAN, C#, CUDA...
- Modern CMake
 - it is called since version 3.0,

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
 - supports generating build-system input files for multiple languages.
 - **C/C++**, FORTRAN, C#, CUDA...
- Modern CMake
 - it is called since version 3.0,
 - or since 2.8.12, to be precise

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
 - supports generating build-system input files for multiple languages.
 - **C/C++**, FORTRAN, C#, CUDA...
- Modern CMake
 - it is called since version 3.0,
 - or since 2.8.12, to be precise
 - is the ***target-centric*** approach
 - Each *target* carries its own build- and usage-requirements.

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
 - supports generating build-system input files for multiple languages.
 - **C/C++**, FORTRAN, C#, CUDA...
- Modern CMake
 - it is called since version 3.0,
 - or since 2.8.12, to be precise
 - is the ***target-centric*** approach
 - Each *target* carries its own build- and usage-requirements.
- Each new version improves CMake

WHAT IS (MODERN) CMAKE?

- CMake
 - is a portable build-system *generator*,
 - generates *input files* for build-systems (Make, Ninja, Visual Studio, ...),
 - supports generating build-system input files for multiple languages.
 - **C/C++**, FORTRAN, C#, CUDA...
- Modern CMake
 - it is called since version 3.0,
 - or since 2.8.12, to be precise
 - is the ***target-centric*** approach
 - Each *target* carries its own build- and usage-requirements.
- Each new version improves CMake
 - and provides new features and **simplifications** .

TWO IMPORTANT TERMS LEARNT

TWO IMPORTANT TERMS LEARNT

BUILD-REQUIREMENTS OF A TARGET

USAGE-REQUIREMENTS OF A TARGET

TWO IMPORTANT TERMS LEARNT

BUILD-REQUIREMENTS OF A TARGET

*"Everything that is needed to (successfully) **build** that target."*

USAGE-REQUIREMENTS OF A TARGET

*"Everything that is needed to (successfully) **use** that target,
as a dependency of another target."*

TWO IMPORTANT TERMS LEARNT

BUILD-REQUIREMENTS OF A TARGET

*"Everything that is needed to (successfully) **build** that target."*

USAGE-REQUIREMENTS OF A TARGET

*"Everything that is needed to (successfully) **use** that target,
as a dependency of another target."*

- source-files
- include search-paths
- pre-processor macros
- link-dependencies
- compiler/linker-options
- compiler/linker-features
 - (e.g. support for a C++-standard)

COMPARISON

build-requirements are set on?

keeping track of
usage-requirements

usage-requirements propagation
from dependency (by using
`target_link_libraries` command)

Traditional CMake

on **environment** (mainly)
e.g. *directory scope*

via (cache-)**variables**

explicit propagation
by hand**

More error-prone!

Modern CMake

on **targets***

via **targets**
(*keep track themselves*)

automatic propagation

Less error-prone!

Allows for more **fine-grained configuration**.

* Or already on dependencies.

** Only paths to library-files are propagated by default.

MODERN CMAKE

SETTING BUILD-REQUIREMENTS VS SETTING USAGE-REQUIREMENTS

```
01 # Adding build-requirements
02
03 target_include_directories( <target> PRIVATE <include-search-dir>... )
04 target_compile_definitions( <target> PRIVATE <macro-definitions>... )
05 target_compile_options(      <target> PRIVATE <compiler-option>... )
06 target_compile_features(    <target> PRIVATE <feature>... )
07 target_sources(            <target> PRIVATE <source-file>... )
08 target_precompile_headers( <target> PRIVATE <header-file>... )
09 target_link_libraries(     <target> PRIVATE <dependency>... )
10 target_link_options(       <target> PRIVATE <linker-option>... )
11 target_link_directories(  <target> PRIVATE <linker-search-dir>... )
```

```
01 # Adding usage-requirements
02
03 target_include_directories( <target> INTERFACE <include-search-dir>... )
04 target_compile_definitions( <target> INTERFACE <macro-definitions>... )
05 target_compile_options(    <target> INTERFACE <compiler-option>... )
06 target_compile_features(  <target> INTERFACE <feature>... )
07 target_sources(           <target> INTERFACE <source-file>... )
08 target_precompile_headers( <target> INTERFACE <header-file>... )
09 target_link_libraries(    <target> INTERFACE <dependency>... )
10 target_link_options(      <target> INTERFACE <linker-option>... )
11 target_link_directories( <target> INTERFACE <linker-search-dir>... )
```

MODERN CMAKE

SETTING BUILD-REQUIREMENTS VS SETTING USAGE-REQUIREMENTS

```
01 # Adding build-requirements
02
03 target_include_directories( <target> PRIVATE <include-search-dir>... )
04 target_compile_definitions( <target> PRIVATE <macro-definitions>... )
05 target_compile_options(      <target> PRIVATE <compiler-option>... )
06 target_compile_features(    <target> PRIVATE <feature>... )
07 target_sources(            <target> PRIVATE <source-file>... )
08 target_precompile_headers( <target> PRIVATE <header-file>... )
09 target_link_libraries(     <target> PRIVATE <dependency>... )
10 target_link_options(       <target> PRIVATE <linker-option>... )
11 target_link_directories(  <target> PRIVATE <linker-search-dir>... )
```

```
01 # Adding usage-requirements
02
03 target_include_directories( <target> INTERFACE <include-search-dir>... )
04 target_compile_definitions( <target> INTERFACE <macro-definitions>... )
05 target_compile_options(    <target> INTERFACE <compiler-option>... )
06 target_compile_features(  <target> INTERFACE <feature>... )
07 target_sources(           <target> INTERFACE <source-file>... )
08 target_precompile_headers( <target> INTERFACE <header-file>... )
09 target_link_libraries(    <target> INTERFACE <dependency>... )
10 target_link_options(      <target> INTERFACE <linker-option>... )
11 target_link_directories( <target> INTERFACE <linker-search-dir>... )
```

MODERN CMAKE

SETTING BUILD-REQUIREMENTS VS SETTING USAGE-REQUIREMENTS

```
01 # Adding build-requirements
02
03 target_include_directories( <target> PRIVATE <include-search-dir>... )
04 target_compile_definitions( <target> PRIVATE <macro-definitions>... )
05 target_compile_options(      <target> PRIVATE <compiler-option>... )
06 target_compile_features(    <target> PRIVATE <feature>... )
07 target_sources(            <target> PRIVATE <source-file>... )
08 target_precompile_headers( <target> PRIVATE <header-file>... )
09 target_link_libraries(     <target> PRIVATE <dependency>... )
10 target_link_options(       <target> PRIVATE <linker-option>... )
11 target_link_directories(  <target> PRIVATE <linker-search-dir>... )
```

```
01 # Adding usage-requirements
02
03 target_include_directories( <target> INTERFACE <include-search-dir>... )
04 target_compile_definitions( <target> INTERFACE <macro-definitions>... )
05 target_compile_options(    <target> INTERFACE <compiler-option>... )
06 target_compile_features(  <target> INTERFACE <feature>... )
07 target_sources(           <target> INTERFACE <source-file>... )
08 target_precompile_headers( <target> INTERFACE <header-file>... )
09 target_link_libraries(    <target> INTERFACE <dependency>... )
10 target_link_options(      <target> INTERFACE <linker-option>... )
11 target_link_directories( <target> INTERFACE <linker-search-dir>... )
```

MODERN CMAKE

SETTING BUILD-REQUIREMENTS VS SETTING USAGE-REQUIREMENTS

```
01 # Adding build-requirements
02
03 target_include_directories( <target> PRIVATE <include-search-dir>... )
04 target_compile_definitions( <target> PRIVATE <macro-definitions>... )

01 # Adding build- and usage-requirements
02
03 target_include_directories( <target> PUBLIC <include-search-dir>... )
04 target_compile_definitions( <target> PUBLIC <macro-definitions>... )
05 target_compile_options(      <target> PUBLIC <compiler-option>... )
06 target_compile_features(     <target> PUBLIC <feature>... )
07 target_sources(              <target> PUBLIC <source-file>... )
08 target_precompile_headers(   <target> PUBLIC <header-file>... )
09 target_link_libraries(       <target> PUBLIC <dependency>... )
10 target_link_options(         <target> PUBLIC <linker-option>... )
11 target_link_directories(    <target> PUBLIC <linker-search-dir>... )

07 target_sources(              <target> INTERFACE <source-file>... )
08 target_precompile_headers(   <target> INTERFACE <header-file>... )
09 target_link_libraries(       <target> INTERFACE <dependency>... )
10 target_link_options(         <target> INTERFACE <linker-option>... )
11 target_link_directories(    <target> INTERFACE <linker-search-dir>... )
```

MODERN CMAKE

SETTING BUILD-REQUIREMENTS VS SETTING USAGE-REQUIREMENTS

```
01 # Adding build-requirements
02
03 target_include_directories( <target> PRIVATE <include-search-dir>... )
04 target_compile_definitions( <target> PRIVATE <macro-definitions>... )
05 target_compile_options(      <target> PRIVATE <compiler-option>... )
06 target_compile_features(    <target> PRIVATE <feature>... )
01 # Adding build- and usage-requirements
02
03 target_include_directories( <target> PUBLIC <include-search-dir>... )
04 target_compile_definitions( <target> PUBLIC <macro-definitions>... )
05 target_compile_options(      <target> PUBLIC <compiler-option>... )
06 target_compile_features(    <target> PUBLIC <feature>... )
07 target_sources(             <target> PUBLIC <source-file>... )
08 target_precompile_headers(  <target> PUBLIC <header-file>... )
09 target_link_libraries(      <target> PUBLIC <dependency>... )
10 target_link_options(        <target> PUBLIC <linker-option>... )
11 target_link_directories(    <target> PUBLIC <linker-search-dir>... )
05 target_compile_options(    <target> INTERFACE <compiler-option>... )
06 target_compile_features(   <target> INTERFACE <feature>... )
07 target_sources(            <target> INTERFACE <source-file>... )
08 target_precompile_headers( <target> INTERFACE <header-file>... )
```

Warning: Although `target_link_libraries` can be used without these keywords, you should **never forget to use these keywords** in Modern CMake!

IMPROVEMENTS AND FIXES TO FEATURES PRESENTED LAST YEAR

SIMPLIFICATIONS TO `target_sources`

- Last year's recommendation:
 - Always use `target_sources` to add all sources.
 - Use `target_sources` to add header-files, too!

```
01 # ./CMakeLists.txt
02
03 add_library( MyTarget SHARED )
04 # Add some sources to target.
05 target_sources( MyTarget
06     PRIVATE    src/A.cpp
07             src/B.cpp
08             headers/B.hpp
09     PUBLIC      ${CMAKE_CURRENT_SOURCE_DIR}/headers/A.hpp
10     INTERFACE   ${CMAKE_CURRENT_SOURCE_DIR}/headers/C.hpp
11 )
```

```
.
├── CMakeLists.txt
├── src/
│   ├── A.cpp
│   └── B.cpp
└── headers/
    ├── A.hpp
    ├── B.hpp
    └── C.hpp
└── subdir/
    ├── CMakeLists.txt
    ├── extra_src/
    │   └── D.cpp
    └── extra_headers/
        └── D.hpp
```

```
01 # ./subdir/CMakeLists.txt
02
03 # Add further sources to target.
04 target_sources( MyTarget
05     PRIVATE    subdir/extra_src/D.cpp
06     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/extra_headers/D.hpp
07 )
```

SIMPLIFICATIONS TO `target_sources`

- Last year's recommendation:
 - Always use `target_sources` to add all sources.
 - Use `target_sources` to add header-files, too!
 - Helps IDEs to show all sources.
 - Might have some positive implications in the future, too.

```
01 # ./CMakeLists.txt
02
03 add_library( MyTarget SHARED )
04 # Add some sources to target.
05 target_sources( MyTarget
06     PRIVATE    src/A.cpp
07                 src/B.cpp
08                 headers/B.hpp
09     PUBLIC     ${CMAKE_CURRENT_SOURCE_DIR}/headers/A.hpp
10     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/headers/C.hpp
11 )
```

```
.
├── CMakeLists.txt
├── src/
│   ├── A.cpp
│   └── B.cpp
└── headers/
    ├── A.hpp
    ├── B.hpp
    └── C.hpp
└── subdir/
    ├── CMakeLists.txt
    ├── extra_src/
    │   └── D.cpp
    └── extra_headers/
        └── D.hpp
```

```
01 # ./subdir/CMakeLists.txt
02
03 # Add further sources to target.
04 target_sources( MyTarget
05     PRIVATE    subdir/extra_src/D.cpp
06     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/extra_headers/D.hpp
07 )
```

SIMPLIFICATIONS TO `target_sources`

- Last year's recommendation:
 - Always use `target_sources` to add all sources.
 - Use `target_sources` to add header-files, too!
 - Helps IDEs to show all sources.
 - Might have some positive implications in the future, too.

```
01 # ./CMakeLists.txt
02
03 add_library( MyTarget SHARED )
04 # Add some sources to target.
05 target_sources( MyTarget
06     PRIVATE    src/A.cpp
07                 src/B.cpp
08                 headers/B.hpp
09     PUBLIC     ${CMAKE_CURRENT_SOURCE_DIR}/headers/A.hpp
10     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/headers/C.hpp
11 )
```

```
.
├── CMakeLists.txt
└── src/
    ├── A.cpp
    └── B.cpp
└── headers/
    ├── A.hpp
    ├── B.hpp
    └── C.hpp
└── subdir/
    ├── CMakeLists.txt
    └── extra_src/
        └── D.cpp
    └── extra_headers/
        └── D.hpp
```

```
01 # ./subdir/CMakeLists.txt
02
03 # Add further sources to target.
04 target_sources( MyTarget
05     PRIVATE    subdir/extra_src/D.cpp
06     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/extra_headers/D.hpp
07 )
```

SIMPLIFICATIONS TO `target_sources`

- Last year's recommendation:
 - Always use `target_sources` to add all sources.
 - Use `target_sources` to add header-files, too!
 - Helps IDEs to show all sources.
 - Might have some positive implications in the future, too.

```
01 # ./CMakeLists.txt
02
03 add_library( MyTarget SHARED )
04 # Add some sources to target.
05 target_sources( MyTarget
06     PRIVATE    src/A.cpp
07                 src/B.cpp
08                 headers/B.hpp
09     PUBLIC     ${CMAKE_CURRENT_SOURCE_DIR}/headers/A.hpp
10     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/headers/C.hpp
11 )
```

```
.
├── CMakeLists.txt
└── src/
    ├── A.cpp
    └── B.cpp
└── headers/
    ├── A.hpp
    ├── B.hpp
    └── C.hpp
└── subdir/
    ├── CMakeLists.txt
    └── extra_src/
        └── D.cpp
    └── extra_headers/
        └── D.hpp
```

```
01 # ./subdir/CMakeLists.txt
02
03 # Add further sources to target.
04 target_sources( MyTarget
05     PRIVATE    subdir/extra_src/D.cpp
06     INTERFACE  ${CMAKE_CURRENT_SOURCE_DIR}/extra_headers/D.hpp
07 )
```

SIMPLIFICATIONS TO `target_sources`

- Last year's recommendation:
 - Always use `target_sources` to add all sources.
 - Use `target_sources` to add header-files, too!
 - Helps IDEs to show all sources.
 - Might have some positive implications in the future, too.
- Simplifications/Fixes with CMake 3.13
 - `target_sources` now correctly interprets relative paths as relative to current `CMAKE_CURRENT_SOURCE_DIR`
 - Relative paths will be converted to absolute paths.

```
01 # ./CMakeLists.txt
02
03 add_library( MyTarget SHARED )
04 # Add some sources to target.
05 target_sources( MyTarget
06     PRIVATE    src/A.cpp
07                 src/B.cpp
08                 headers/B.hpp
09     PUBLIC     headers/A.hpp
10     INTERFACE  headers/C.hpp
11 )
```

```
.
├── CMakeLists.txt
└── src/
    ├── A.cpp
    └── B.cpp
└── headers/
    ├── A.hpp
    ├── B.hpp
    └── C.hpp
└── subdir/
    ├── CMakeLists.txt
    └── extra_src/
        └── D.cpp
    └── extra_headers/
        └── D.hpp
```

```
01 # ./subdir/CMakeLists.txt
02
03 # Add further sources to target.
04 target_sources( MyTarget
05     PRIVATE    extra_src/D.cpp
06     INTERFACE  extra_headers/D.hpp
07 )
```

OBJECT LIBRARIES

OBJECT LIBRARIES

OBJECT libraries are like any other CMake targets.

OBJECT LIBRARIES

OBJECT libraries are like any other CMake targets...
except when they are not.

OBJECT libraries are like any other CMake targets...
except when they are not.

```
01 add_library( obj OBJECT )
02 target_sources( obj
03     PRIVATE src/source1.cpp
04         src/source2.cpp
05         src/source3.cpp
06 )
07 target_include_directories( obj INTERFACE ./headers )
08 target_compile_definitions( obj INTERFACE "IS_EXAMPLE=1" )
```

- OBJECT library obj carries
 - **usage-requirements**
 - *include-search-path* (`./headers`)
 - *preprocessor-definition* (`IS_EXAMPLE=1`)
 - **object files**
 - generated from its private sources

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib PRIVATE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib INTERFACE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib PUBLIC obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
02 ...
03 add_library( lib SHARED )
04 target_sources( lib PRIVATE src.cpp )
05 target_link_libraries( lib PRIVATE obj )
06
07 add_executable( exe )
08 target_sources( exe PRIVATE main.cpp )
09 target_link_libraries( exe PRIVATE lib )

01 add_library( obj OBJECT )
02 ...
03 add_library( lib SHARED )
04 target_sources( lib PRIVATE src.cpp )
05 target_link_libraries( lib INTERFACE obj )
06
07 add_executable( exe )
08 target_sources( exe PRIVATE main.cpp )
09 target_link_libraries( exe PRIVATE lib )
```

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib PRIVATE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib INTERFACE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib PUBLIC obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

usage-requirements are propagated as always:

obj → lib ↗ exe

obj (→ lib) → exe

obj → lib → exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib PRIVATE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib INTERFACE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( lib PUBLIC obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE lib )
```

usage-requirements are propagated as always:

obj → lib ↗ exe

obj (→ lib) → exe

obj → lib → exe

object files are propagated differently:

obj → lib ↗ exe

obj ↗ lib ↗ exe

obj → lib ↗ exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries on *left*- and *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 PRIVATE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 INTERFACE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 PUBLIC obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries on *left*- and *right-hand-side* of target_link_libraries

01 add_library(obj OBJECT)	01 add_library(obj OBJECT)	01 add_library(obj OBJECT)
09 ...	09 ...	09 ...
10 add_library(obj2 OBJECT)	10 add_library(obj2 OBJECT)	10 add_library(obj2 OBJECT)
11 target_sources(obj2 PRIVATE src.cpp)	11 target_sources(obj2 PRIVATE src.cpp)	11 target_sources(obj2 PRIVATE src.cpp)
12 target_link_libraries(obj2 PRIVATE obj)	12 target_link_libraries(obj2 INTERFACE obj)	12 target_link_libraries(obj2 PUBLIC obj)
13	13	13
14 add_executable(exe)	14 add_executable(exe)	14 add_executable(exe)
15 target_sources(exe PRIVATE main.cpp)	15 target_sources(exe PRIVATE main.cpp)	15 target_sources(exe PRIVATE main.cpp)
16 target_link_libraries(exe PRIVATE obj2)	16 target_link_libraries(exe PRIVATE obj2)	16 target_link_libraries(exe PRIVATE obj2)

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries on *left-* and *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 PRIVATE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 INTERFACE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 PUBLIC obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

usage-requirements are propagated as always:

obj → obj2 $\not\rightarrow$ exe

obj (→ obj2) → exe

obj → obj2 → exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries on *left*- and *right-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 PRIVATE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 INTERFACE obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( obj2 OBJECT )
11 target_sources( obj2 PRIVATE src.cpp )
12 target_link_libraries( obj2 PUBLIC obj )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj2 )
```

usage-requirements are propagated as always:

obj → obj2 $\not\rightarrow$ exe

obj (→ obj2) → exe

obj → obj2 → exe

object files are never propagated to/through other OBJECT libraries

obj $\not\rightarrow$ obj2 $\not\rightarrow$ exe

obj $\not\rightarrow$ obj2 $\not\rightarrow$ exe

obj $\not\rightarrow$ obj2 $\not\rightarrow$ exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *left-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PRIVATE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj INTERFACE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PUBLIC lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *left-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
02 ...
03 add_library( lib SHARED )
04 target_sources( lib PRIVATE src.cpp )
05 target_link_libraries( obj PRIVATE lib )
06 ...
07 add_executable( exe )
08 target_sources( exe PRIVATE main.cpp )
09 target_link_libraries( exe PRIVATE obj )
10 ...
11 add_library( lib SHARED )
12 target_sources( lib PRIVATE src.cpp )
13 target_link_libraries( obj INTERFACE lib )
14 ...
15 add_executable( exe )
16 target_sources( exe PRIVATE main.cpp )
17 target_link_libraries( exe PRIVATE obj )
18 ...
19 add_library( obj OBJECT )
20 ...
21 add_library( lib SHARED )
22 target_sources( lib PRIVATE src.cpp )
23 target_link_libraries( obj PUBLIC lib )
24 ...
25 add_executable( exe )
26 target_sources( exe PRIVATE main.cpp )
27 target_link_libraries( exe PRIVATE obj )
```

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *left-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PRIVATE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj INTERFACE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PUBLIC lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

usage-requirements are propagated as always:

lib → obj ↗ exe

lib (→ obj) → exe

lib → obj → exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *left-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PRIVATE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj INTERFACE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PUBLIC lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

usage-requirements are propagated as always:

lib → obj ↗ exe

lib (→ obj) → exe

lib → obj → exe

link-dependency to lib are propagated differently:

lib ↗ obj ↗ exe

lib (→ obj) → exe

lib (→ obj) → exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES

OBJECT libraries only on *left-hand-side* of target_link_libraries

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PRIVATE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj INTERFACE lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

```
01 add_library( obj OBJECT )
09 ...
10 add_library( lib SHARED )
11 target_sources( lib PRIVATE src.cpp )
12 target_link_libraries( obj PUBLIC lib )
13
14 add_executable( exe )
15 target_sources( exe PRIVATE main.cpp )
16 target_link_libraries( exe PRIVATE obj )
```

usage-requirements are propagated as always:

lib → obj ↗ exe

lib (→ obj) → exe

lib → obj → exe

link-dependency to lib are propagated differently:

lib ↗ obj ↗ exe

lib (→ obj) → exe

lib (→ obj) → exe

link-dependency propagation modified/fixed in CMake 3.14:

lib (→ obj) → exe

lib (→ obj) → exe

lib (→ obj) → exe

OBJECT LIBRARIES

PROPAGATION OF USAGE-REQUIREMENTS / OBJECT FILES / LINK-DEPENDENCIES

	PRIVATE	INTERFACE	PUBLIC
usage-requirements	$obj \rightarrow lib \not\rightarrow exe$	$obj (\rightarrow lib) \rightarrow exe$	$obj \rightarrow lib \rightarrow exe$
usage-requirements	$obj \rightarrow obj2 \not\rightarrow exe$	$obj (\rightarrow obj2) \rightarrow exe$	$obj \rightarrow obj2 \rightarrow exe$
usage-requirements	$lib \rightarrow obj \not\rightarrow exe$	$lib (\rightarrow obj) \rightarrow exe$	$lib \rightarrow obj \rightarrow exe$
object files	$obj \rightarrow lib \not\rightarrow exe$	$obj \not\rightarrow lib \not\rightarrow exe$	$obj \rightarrow lib \not\rightarrow exe$
object files	$obj \not\rightarrow obj2 \not\rightarrow exe$	$obj \not\rightarrow obj2 \not\rightarrow exe$	$obj \not\rightarrow obj2 \not\rightarrow exe$
link-dependencies	$lib (\rightarrow obj) \rightarrow exe$	$lib (\rightarrow obj) \rightarrow exe$	$lib (\rightarrow obj) \rightarrow exe$

AND NOW FOR SOMETHING



COMPLETELY DIFFERENT

source: <https://rich1698.wordpress.com/2018/10/05/monty-pythons-flying-circus>



source: <https://rich1698.wordpress.com/2018/10/05/monty-pythons-flying-circus>

OK, not really different...

LETS START A NEW PROJECT

BEGINNING OF EACH **CMakeLists.txt**

cmake_minimum_required

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
```

BEGINNING OF EACH **CMakeLists.txt**

cmake_minimum_required

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
```

BEGINNING OF EACH **CMakeLists.txt**

cmake_minimum_required

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
```

- Call *cmake_minimum_required*
 - *required*: at begin of *top-level CMakeLists.txt* file.
 - *easier*: at begin of *all CMakeLists.txt* files.

BEGINNING OF EACH `CMakeLists.txt`

`cmake_minimum_required`

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
```

- Call `cmake_minimum_required`
 - *required*: at begin of *top-level* `CMakeLists.txt` file.
 - *easier*: at begin of *all* `CMakeLists.txt` files.
- Sets CMake *policies* to defaults of specific CMake version.
 - `cmake_policy` allows to modify policies again.
(*Policy-scopes* exist, too.)

BEGINNING OF EACH `CMakeLists.txt`

`cmake_minimum_required`

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
```

- Call `cmake_minimum_required`
 - *required*: at begin of *top-level* `CMakeLists.txt` file.
 - *easier*: at begin of *all* `CMakeLists.txt` files.
- Sets CMake *policies* to defaults of specific CMake version.
 - `cmake_policy` allows to modify policies again.
(*Policy-scopes* exist, too.)
- The *version range* `<min-version>...<max-version>` syntax was introduced in 3.12, but is backwards-compatible.

cmake_minimum_required

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
```

- Call *cmake_minimum_required*
 - *required*: at begin of *top-level* CMakeLists.txt file.
 - *easier*: at begin of *all* CMakeLists.txt files.
- Sets CMake *policies* to defaults of specific CMake version.
 - *cmake_policy* allows to modify policies again.
(*Policy-scopes* exist, too.)
- The *version range* <min-version>...<max-version> syntax was introduced in 3.12, but is backwards-compatible.
 - **Recommendation: At least use version 3.15 as minimal version!**
This will allow you to use the shown features.

BEGINNING OF EACH **CMakeLists.txt**

project

1. Define the required CMake-version.

2. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09             VERSION 1.2.3
10             DESCRIPTION "Description of project"
11             HOMEPAGE_URL "https://www.example.com"
12             LANGUAGES C CXX CUDA )
```

BEGINNING OF EACH **CMakeLists.txt**

project

1. Define the required CMake-version.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
```

2. Make this **CMakeLists.txt** file a new project.

```
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09             VERSION 1.2.3
10             DESCRIPTION "Description of project"
11             HOMEPAGE_URL "https://www.example.com"
12             LANGUAGES C CXX CUDA )
```

BEGINNING OF EACH **CMakeLists.txt**

project

1. Define the required CMake-version.

2. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09             VERSION 1.2.3
10             DESCRIPTION "Description of project"
11             HOMEPAGE_URL "https://www.example.com"
12             LANGUAGES C CXX CUDA )
```

- Call *after* `cmake_minimum_required`
 - but as early as possible.

BEGINNING OF EACH **CMakeLists.txt**

project

1. Define the required CMake-version.

2. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09             VERSION 1.2.3
10             DESCRIPTION "Description of project"
11             HOMEPAGE_URL "https://www.example.com"
12             LANGUAGES C CXX CUDA )
```

- Call *after `cmake_minimum_required`*
 - but as early as possible.
- Sets variables containing: project-name, version etc.

project

1. Define the required CMake-version.

2. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09             VERSION 1.2.3
10             DESCRIPTION "Description of project"
11             HOMEPAGE_URL "https://www.example.com"
12             LANGUAGES C CXX CUDA )
```

- Call *after* `cmake_minimum_required`
 - but as early as possible.
- Sets variables containing: project-name, version etc.
- Default values for `LANGUAGES`: `C` and `CXX`
 - Other values: `FORTRAN`, `CUDA`, `CSharp`, `ASM`, `Java` (lol) ...

BEGINNING OF EACH **CMakeLists.txt**

project

1. Define the required CMake-version.

2. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09             VERSION 1.2.3
10             DESCRIPTION "Description of project"
11             HOMEPAGE_URL "https://www.example.com"
12             LANGUAGES C CXX CUDA )
```

- Call *after* `cmake_minimum_required`
 - but as early as possible.
- Sets variables containing: project-name, version etc.
- Default values for `LANGUAGES`: `C` and `CXX`
 - Other values: `FORTRAN`, `CUDA`, `CSharp`, `ASM`, `Java` (lol) ...

include A FILE WITH META-INFOS

1. Define the required CMake-version.
2. Include a (generated) file with project settings.
3. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in" )
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09           VERSION ${project_version}
10           DESCRIPTION ${project_description}
11           HOMEPAGE_URL ${project_homepage}
12           LANGUAGES C CXX CUDA )
```

include A FILE WITH META-INFOS

1. Define the required CMake-version.
2. **Include a (generated) file with project settings.**
3. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in"
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )
```

include A FILE WITH META-INFOS

1. Define the required CMake-version.
2. **Include a (generated) file with project settings.**
3. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in" )
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09           VERSION ${project_version}
10           DESCRIPTION ${project_description}
11           HOMEPAGE_URL ${project_homepage}
12           LANGUAGES C CXX CUDA )
```

- Loads variables from a file that shall be used in *project* command.

```
01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )
```

include A FILE WITH META-INFOS

1. Define the required CMake-version.
2. **Include a (generated) file with project settings.**
3. Make this `CMakeLists.txt` file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in" )
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09           VERSION ${project_version}
10           DESCRIPTION ${project_description}
11           HOMEPAGE_URL ${project_homepage}
12           LANGUAGES C CXX CUDA )
```

- Loads variables from a file that shall be used in `project` command.
 - **In this example:** located in the same directory as `CMakeLists.txt`

```
01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )
```

include A FILE WITH META-INFOS

1. Define the required CMake-version.
2. **Include a (generated) file with project settings.**
3. Make this **CMakeLists.txt** file a new project.

```

01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in" )
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09           VERSION ${project_version}
10           DESCRIPTION ${project_description}
11           HOMEPAGE_URL ${project_homepage}
12           LANGUAGES C CXX CUDA )

```

- Loads variables from a file that shall be used in *project* command.
 - **In this example:** located in the same directory as **CMakeLists.txt**
- Each **CMakeLists.txt** should load its own file.

```

01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )

```

include A FILE WITH META-INFOS

1. Define the required CMake-version.
2. **Include a (generated) file with project settings.**
3. Make this **CMakeLists.txt** file a new project.

```

01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in" )
06
07 # Define a project for the current CMakeLists.txt.
08 project( MyProject
09           VERSION ${project_version}
10           DESCRIPTION ${project_description}
11           HOMEPAGE_URL ${project_homepage}
12           LANGUAGES C CXX CUDA )

```

- Loads variables from a file that shall be used in *project* command.
 - **In this example:** located in the same directory as **CMakeLists.txt**
- Each **CMakeLists.txt** should load its own file.
 - However, that is tedious, hard to remember and too much boiler-plate.

```

01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )

```

CMAKE_PROJECT_INCLUDE_BEFORE

1. Define the required CMake-version.
2. Include a *common file* for all projects.
3. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )
```

CMAKE_PROJECT_INCLUDE_BEFORE

1. Define the required CMake-version.
2. **Include a *common file* for all projects.**
3. Make this **CMakeLists.txt** file a new project.

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in"
07     # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )
```

CMAKE_PROJECT_INCLUDE_BEFORE

1. Define the required CMake-version.
2. **Include a *common file* for all projects.**
3. Make this **CMakeLists.txt** file a new project.

```

01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )

```

- The referenced file is automatically included directly before **each** *project* command

```

01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )

```

CMAKE_PROJECT_INCLUDE_BEFORE

1. Define the required CMake-version.
2. **Include a *common file* for all projects.**
3. Make this **CMakeLists.txt** file a new project.

```

01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )

```

- The referenced file is automatically included directly before **each *project* command**
- and should include each project's meta-info

```

01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )

01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )

```

CMAKE_PROJECT_INCLUDE_BEFORE

1. Define the required CMake-version.
2. **Include a *common file* for all projects.**
3. Make this **CMakeLists.txt** file a new project.

```

01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )

```

- The referenced file is automatically included directly before **each *project* command**
- and should include each project's meta-info, which is relative to the *current CMakeLists.txt file*.
 - Use **CMAKE_CURRENT_SOURCE_DIR** instead of **CMAKE_CURRENT_LIST_DIR**!

```

01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )

01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )

```

CMakeLists.txt COMPARISON

SUB-LEVEL VS TOP-LEVEL

```
01 # subdirectory/CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06
07 # Define a project for the current CMakeLists.txt.
08 project( MySubProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES CXX )
```



```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )
```

CMakeLists.txt COMPARISON

SUB-LEVEL VS TOP-LEVEL

```
01 # subdirectory/CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06     set( CMAKE_PROJECT_INCLUDE_BEFORE
07         "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
08 # Define a project for the current CMakeLists.txt.
09 project( MySubProject
10         VERSION ${project_version}
11         DESCRIPTION ${project_description}
12         HOMEPAGE_URL ${project_homepage}
13         LANGUAGES CXX )
```



```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09         VERSION ${project_version}
10         DESCRIPTION ${project_description}
11         HOMEPAGE_URL ${project_homepage}
12         LANGUAGES C CXX CUDA )
```

```
01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )
```



```
01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )
```

CMakeLists.txt COMPARISON

SUB-LEVEL VS TOP-LEVEL

```
01 # subdirectory/CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05
06 # Define a project for the current CMakeLists.txt.
07 project( MySubProject
08     VERSION ${project_version}
09     DESCRIPTION ${project_description}
10     HOMEPAGE_URL ${project_homepage}
11     LANGUAGES CXX )
```



```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.15...3.17 )
04
05 set( CMAKE_PROJECT_INCLUDE_BEFORE
06     "${CMAKE_CURRENT_LIST_DIR}/common-project-include.in" )
07 # Define a project for the current CMakeLists.txt.
08 project( MyRootProject
09     VERSION ${project_version}
10     DESCRIPTION ${project_description}
11     HOMEPAGE_URL ${project_homepage}
12     LANGUAGES C CXX CUDA )
```

```
01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )
```



```
01 # common-project-info.in
02
03 include( "${CMAKE_CURRENT_SOURCE_DIR}/project-meta-info.in" )
```

```
01 # subdirectory/project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 0.5.8 )
05 # The description of this project.
06 set( project_description "Description of sub-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.org" )
```



```
01 # project-meta-info.in
02
03 # The version number of this project.
04 set( project_version 1.2.3 )
05 # The description of this project.
06 set( project_description "Description of root-project" )
07 # The homepage of this project.
08 set( project_homepage "https://www.example.com" )
```

FINDING EXTERNAL DEPENDENCY

Boost

FINDING EXTERNAL DEPENDENCY - *Boost*

LAST YEAR'S RECOMMENDATION

From a subdirectory's `CMakeLists.txt` file:

- Use `find_package` to locate *Boost*!

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.69.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14               MODULE
15               REQUIRED COMPONENTS program_options
16                           graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

LAST YEAR'S RECOMMENDATION

From a subdirectory's `CMakeLists.txt` file:

- Use `find_package` to locate *Boost*!
 - If found, promote `IMPORTED` targets to global scope.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.69.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   MODULE
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

LAST YEAR'S RECOMMENDATION

From a subdirectory's `CMakeLists.txt` file:

- Use `find_package` to locate *Boost*!
 - If found, promote `IMPORTED` targets to global scope.
- Uses `find_package`'s **Module** mode and the `FindBoost.cmake` module that comes with CMake.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.69.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   MODULE
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

LAST YEAR'S RECOMMENDATION

From a subdirectory's `CMakeLists.txt` file:

- Use `find_package` to locate *Boost*!
 - If found, promote `IMPORTED` targets to global scope.
- Uses `find_package`'s **Module** mode and the `FindBoost.cmake` module that comes with CMake.
 - For *Boost* versions newer than `FindBoost.cmake` version, the variable `Boost_ADDITIONAL_VERSIONS` has to contain the additional version(s).

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.69.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   MODULE
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

LAST YEAR'S RECOMMENDATION

From a subdirectory's `CMakeLists.txt` file:

- Use `find_package` to locate *Boost*!
 - If found, promote `IMPORTED` targets to global scope.
- Uses `find_package`'s **Module** mode and the `FindBoost.cmake` module that comes with CMake.
 - For *Boost* versions newer than `FindBoost.cmake` version, the variable `Boost_ADDITIONAL_VERSIONS` has to contain the additional version(s).
 - This is error-prone!
 - Dependencies might be wrong.
 - `IMPORTED` targets for new *Boost* libraries will probably not be created.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.69.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   MODULE
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Starting with version 1.70.0 *Boost* provides its own
BoostConfig.cmake file:

- Use `find_package` in **Config** mode to locate *Boost*!

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Starting with version 1.70.0 *Boost* provides its own `BoostConfig.cmake` file:

- Use `find_package` in **Config** mode to locate *Boost*!
- Searches for the `BoostConfig.cmake` script and creates **IMPORTED** targets from it.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::boost
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Starting with version 1.70.0 *Boost* provides its own `BoostConfig.cmake` file:

- Use `find_package` in ***Config*** mode to locate *Boost*!
- Searches for the `BoostConfig.cmake` script and creates **IMPORTED** targets from it.
 - Target `Boost::boost` was renamed to `Boost::headers` (but an alias is still available).

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10 set( Boost_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Starting with version 1.70.0 *Boost* provides its own `BoostConfig.cmake` file:

- Use `find_package` in ***Config*** mode to locate *Boost*!
- Searches for the `BoostConfig.cmake` script and creates **IMPORTED** targets from it.
 - Target `Boost::boost` was renamed to `Boost::headers` (but an alias is still available).
 - The variable `Boost_ADDITIONAL_VERSIONS` is no longer needed.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Starting with version 1.70.0 *Boost* provides its own `BoostConfig.cmake` file:

- Use `find_package` in ***Config*** mode to locate *Boost*!
- Searches for the `BoostConfig.cmake` script and creates **IMPORTED** targets from it.
 - Target `Boost::boost` was renamed to `Boost::headers` (but an alias is still available).
 - The variable `Boost_ADDITIONAL_VERSIONS` is no longer needed.
 - \Rightarrow **Save with all versions of CMake!**

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.70.0$

Starting with version 1.70.0 *Boost* provides its own `BoostConfig.cmake` file:

- Use `find_package` in ***Config*** mode to locate *Boost*!
- Searches for the `BoostConfig.cmake` script and creates **IMPORTED** targets from it.
 - Target `Boost::boost` was renamed to `Boost::headers` (but an alias is still available).
 - The variable `Boost_ADDITIONAL_VERSIONS` is no longer needed.
 - \Rightarrow **Save with all versions of CMake!**
 - ...with all versions $\geq 2.8.8$, to be precise.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Common for all *Boost* versions:

- You must explicitly specify the components!

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER            "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS program_options
16   graph )
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.70.0

Common for all *Boost* versions:

- You must explicitly specify the components!
- Omitting any component only looks for ***header-only*** *Boost* libraries.
⇒ `Boost::headers`

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED )
16
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22
23
24   PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.70.0$

Common for all *Boost* versions:

- You must explicitly specify the components!
- Omitting any component only looks for ***header-only*** *Boost* libraries.
⇒ `Boost::headers`

Recommendation for *Boost* versions $\geq 1.70.0$:

- List `headers` component explicitly!
 - In particular, if only ***header-only*** libraries are need.

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS headers )
16
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22
23
24   PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.70.0$

Common for all *Boost* versions:

- You must explicitly specify the components!
- Omitting any component only looks for ***header-only*** *Boost* libraries.

\Rightarrow `Boost::headers`

Recommendation for *Boost* versions $\geq 1.70.0$:

- List `headers` component explicitly!
 - In particular, if only ***header-only*** libraries are need.
 - ***This is more future-proof!***

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.70.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS FALSE )
07 set( Boost_USE_MULTITHREADED TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS headers )
16
17
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22
23
24   PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS \geq 1.73.0

Common for all *Boost* versions:

- **You must explicitly specify the components!**

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.73.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS headers
16   program_options
17   graph )
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.73.0$

Common for all *Boost* versions:

- **You must explicitly specify the components!**

What if you want to import all of *Boost*?

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.73.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS headers
16   program_options
17   graph )
18
19 # Make found targets globally available.
20 if ( Boost_FOUND )
21   set_target_properties( Boost::headers
22                         Boost::program_options
23                         Boost::graph
24                         PROPERTIES IMPORTED_GLOBAL TRUE )
25 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.73.0$

Common for all *Boost* versions:

- **You must explicitly specify the components!**

What if you want to import all of *Boost*?



```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.73.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME  FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS headers
16   atomic
17   chrono
18   container
19   context
20   contract
21   coroutine
22   data_time
23   exception
24   fiber
25   fiber_numa
26   filesystem
27   graph
28   graph_parallel
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.73.0$

Common for all *Boost* versions:

- You must explicitly specify the components!

What if you want to import all of *Boost*?



- *Boost 1.73.0* (and newer) to the rescue:

ALL component

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.73.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED COMPONENTS ALL )
16
17
18 # Make found targets globally available.
19 if ( Boost_FOUND )
20   set_target_properties( ${Boost_ALL_TARGETS}
21     PROPERTIES IMPORTED_GLOBAL TRUE )
22 endif ()
```

FINDING EXTERNAL DEPENDENCY - *Boost*

NEWER BOOST VERSIONS $\geq 1.73.0$

Common for all *Boost* versions:

- You must explicitly specify the components!

What if you want to import all of *Boost*?



- *Boost 1.73.0* (and newer) to the rescue:

ALL component

- If REQUIRED keyword is given, the COMPONENTS keyword can be omitted.

⇒ looks quite nice with ALL component

```
01 # ./external/boost/CMakeLists.txt
02
03 set( BOOST_VERSION 1.73.0 )
04
05 # Settings for finding correct Boost libraries.
06 set( Boost_USE_STATIC_LIBS      FALSE )
07 set( Boost_USE_MULTITHREADED   TRUE )
08 set( Boost_USE_STATIC_RUNTIME FALSE )
09 set( Boost_COMPILER             "-gcc8" )
10
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT
14   CONFIG
15   REQUIRED ALL )
16
17
18 # Make found targets globally available.
19 if ( Boost_FOUND )
20   set_target_properties( ${Boost_ALL_TARGETS}
21                         PROPERTIES IMPORTED_GLOBAL TRUE )
22 endif ()
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

USE-CASE

- Sometimes, using pre-built dependencies is not feasible.

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

USE-CASE

- Sometimes, using pre-built dependencies is not feasible.
 - The dependency needs **custom compiler flags**, provided by my project, or
 - it is **not general enough** to be of use for more than my project,
 - ...

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

USE-CASE

- Sometimes, using pre-built dependencies is not feasible.
 - The dependency needs **custom compiler flags**, provided by my project, or
 - it is **not general enough** to be of use for more than my project,
 - ...
- Solution: Built the dependency together with your project!

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

USE-CASE

- Sometimes, using pre-built dependencies is not feasible.
 - The dependency needs **custom compiler flags**, provided by my project, or
 - it is **not general enough** to be of use for more than my project,
 - ...
- Solution: Built the dependency together with your project!
 - Then I need to checkout the code when building!? 😞

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

USE-CASE

- Sometimes, using pre-built dependencies is not feasible.
 - The dependency needs **custom compiler flags**, provided by my project, or
 - it is **not general enough** to be of use for more than my project,
 - ...
- Solution: Built the dependency together with your project!
 - Then I need to checkout the code when building!? 😞
 - But I want to use its CMake targets before, when configuring my project!!! 🤯

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

USE-CASE

- Sometimes, using pre-built dependencies is not feasible.
 - The dependency needs **custom compiler flags**, provided by my project, or
 - it is **not general enough** to be of use for more than my project,
 - ...
- Solution: Built the dependency together with your project!
 - Then I need to checkout the code when building!? 😞
 - But I want to use its CMake targets before, when configuring my project!!! 🤯

Introducing:
FetchContent CMake module

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

ONE IMPORTANT CONSTRAINT

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

ONE IMPORTANT CONSTRAINT

FetchContent only works with dependencies that themselves use CMake to build!

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETEST*

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETEST*

1. Load *FetchContent* module

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
05 # Load FetchContent module.
06 include( FetchContent )
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
05 # Load FetchContent module.
06 include( FetchContent )
07
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

2. Need to tell *FetchContent*

- what code to fetch for building
- and where to find it.
- ⇒ `FetchContent_Declare`

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
05 # Load FetchContent module.
06 include( FetchContent )
08 # Declare GoogleTest as the content to fetch.
09 FetchContent_Declare(
10   googletest
11   GIT_REPOSITORY https://github.com/google/googletest.g
12   GIT_TAG         release-1.8.0
13 )
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

2. Need to tell *FetchContent*

- what code to fetch for building
- and where to find it.
- ⇒ `FetchContent_Declare`

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
05 # Load FetchContent module.
06 include( FetchContent )
07
08 # Declare GoogleTest as the content to fetch.
09 FetchContent_Declare(
10     googletest
11     GIT_REPOSITORY https://github.com/google/googletest.git
12     GIT_TAG        release-1.8.0
13 )
14
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

2. Need to tell *FetchContent*

- what code to fetch for building
- and where to find it.
- ⇒ `FetchContent_Declare`

3. Fetch the content

- making its `CMakeLists.txt` script available (via `add_subdirectory`).
- ⇒ `FetchContent_MakeAvailable`

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
05 # Load FetchContent module.
06 include( FetchContent )
07
08 # Declare GoogleTest as the content to fetch.
09 FetchContent_Declare(
10     googletest
11     GIT_REPOSITORY https://github.com/google/googletest.git
12     GIT_TAG         release-1.8.0
13 )
14
15 # Fetch GoogleTest and make build scripts available.
16 FetchContent_MakeAvailable( googletest )
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETTEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

2. Need to tell *FetchContent*

- what code to fetch for building
- and where to find it.
- ⇒ `FetchContent_Declare`

3. Fetch the content

- making its `CMakeLists.txt` script available
(via `add_subdirectory`).
- ⇒ `FetchContent_MakeAvailable`
 - General case
which does not always work without modifications!

```
01 # ./myproject/CMakeLists.txt
02 ...
03 ...
04 ...
05 # Load FetchContent module.
06 include( FetchContent )
07 ...
08 # Declare GoogleTest as the content to fetch.
09 FetchContent_Declare(
10     googletest
11     GIT_REPOSITORY https://github.com/google/googletest.git
12     GIT_TAG        release-1.8.0
13 )
14 ...
15 # Fetch GoogleTest and make build scripts available.
16 FetchContent_MakeAvailable( googletest )
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETTEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

2. Need to tell *FetchContent*

- what code to fetch for building
- and where to find it.
- ⇒ `FetchContent_Declare`

3. Fetch the content

- making its `CMakeLists.txt` script available (via `add_subdirectory`).
- ⇒ `FetchContent_MakeAvailable`
 - General case
which does not always work without modifications!

```
01 # ./myproject/CMakeLists.txt
02
03 ...
04
05 # Load FetchContent module.
06 include( FetchContent )
07
08 # Declare GoogleTest as the content to fetch.
09 FetchContent_Declare(
10     googletest
11     GIT_REPOSITORY https://github.com/google/googletest.git
12     GIT_TAG        release-1.8.0
13 )
14
15 # Fetch GoogleTest and make build scripts available.
16 if (NOT googletest_POPULATED)
17     # Fetch the content using previously declared details.
18     FetchContent_Populate( googletest )
19
20     # Custom policies, variables and modifications go here.
21     # ...
22
23     # Bring the populated content into the build.
24     add_subdirectory( ${googletest_SOURCE_DIR}
25                       ${googletest_BINARY_DIR} )
26 endif()
```

BUILDING EXTERNAL DEPENDENCIES WITH *FETCHCONTENT*

DEMONSTRATION WITH *GOOGLETTEST*

1. Load *FetchContent* module

- Bundled with CMake since version 3.11.

2. Need to tell *FetchContent*

- what code to fetch for building
- and where to find it.
- ⇒ `FetchContent_Declare`

3. Fetch the content

- making its `CMakeLists.txt` script available (via `add_subdirectory`).
- ⇒ `FetchContent_MakeAvailable`
 - General case
which does not always work without modifications!

```
01 # ./myproject/CMakeLists.txt
02 ...
03 ...
04 ...
05 # Load FetchContent module.
06 include( FetchContent )
07 ...
08 # Declare GoogleTest as the content to fetch.
09 FetchContent_Declare(
10     googletest
11     GIT_REPOSITORY https://github.com/google/googletest.git
12     GIT_TAG        release-1.8.0
15 # Fetch GoogleTest and make build scripts available.
16 if (NOT googletest_POPULATED)
17     # Fetch the content using previously declared details
18     FetchContent_Populate( googletest )
19 ...
20 # Custom policies, variables and modifications go here
21 ...
22 ...
23 # Bring the populated content into the build.
24 add_subdirectory( ${googletest_SOURCE_DIR}
25                   ${googletest_BINARY_DIR} )
26 endif()
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

The `CMakeLists.txt` files of *GoogleTest* 1.8.0 are really old, and print this annoying warning.

```
01 CMake Warning (dev) at ../../build-dir/_deps/googletest-src/CMakeLists.txt:3 (project):
02   Policy CMP0048 is not set: project() command manages VERSION variables.
03   Run "cmake --help-policy CMP0048" for policy details. Use the cmake_policy
04   command to set the policy and suppress this warning.
05
06   The following variable(s) would be set to empty:
07
08     PROJECT_VERSION
09     PROJECT_VERSION_MAJOR
10     PROJECT_VERSION_MINOR
11     PROJECT_VERSION_PATCH
12 This warning is for project developers. Use -Wno-dev to suppress it.
13
14 CMake Warning (dev) at ../../build-dir/_deps/googletest-src/googletest/CMakeLists.txt:40 (project):
15   Policy CMP0048 is not set: project() command manages VERSION variables.
16 ...
17 This warning is for project developers. Use -Wno-dev to suppress it.
18
19 CMake Warning (dev) at ../../build-dir/_deps/googletest-src/googletest/CMakeLists.txt:47 (project):
20   Policy CMP0048 is not set: project() command manages VERSION variables.
21 ...
22 This warning is for project developers. Use -Wno-dev to suppress it.
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

The `CMakeLists.txt` files of *GoogleTest* 1.8.0 are really old, and print this annoying warning.

```
01 CMake Warning (dev) at ../../build-dir/_deps/googletest-src/CMakeLists.txt:3 (project):
02 Policy CMP0048 is not set: project() command manages VERSION variables.
03 Run "cmake --help-policy CMP0048" for policy details. Use the cmake_policy
04 command to set the policy and suppress this warning.
05
06 The following variable(s) would be set to empty:
07
08 PROJECT_VERSION
09 PROJECT_VERSION_MAJOR
10 PROJECT_VERSION_MINOR
11 PROJECT_VERSION_PATCH
12 This warning is for project developers. Use -Wno-dev to suppress it.
13
14 CMake Warning (dev) at ../../build-dir/_deps/googletest-src/googletest/CMakeLists.txt:40 (project):
15 Policy CMP0048 is not set: project() command manages VERSION variables.
16 ...
17 This warning is for project developers. Use -Wno-dev to suppress it.
18
19 CMake Warning (dev) at ../../build-dir/_deps/googletest-src/googletest/CMakeLists.txt:47 (project):
20 Policy CMP0048 is not set: project() command manages VERSION variables.
21 ...
22 This warning is for project developers. Use -Wno-dev to suppress it.
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.  
11 ...  
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.  
11 ...  
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- But how to set CMake command-line option `-Wno-dev` through `FetchContent`?

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.  
11 ...  
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- But how to set CMake command-line option `-Wno-dev` through **FetchContent**?
 - **That's not possible!** 

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- But how to set CMake command-line option `-Wno-dev` through `FetchContent`?
 - **That's not possible!** 
- OK, then let's set the CMake policy `CMP0048`:

```
15 ...
16 # Try to set policy CMP0048 for GoogleTest project.
17 cmake_policy( SET CMP0048 NEW )
18
19 # Fetch GoogleTest and make build scripts available.
20 FetchContent_MakeAvailable( googletest )
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- But how to set CMake command-line option `-Wno-dev` through `FetchContent`?
 - **That's not possible!** 
- OK, then let's set the CMake policy `CMP0048`:

```
15 ...
16 # Try to set policy CMP0048 for GoogleTest project.
17 cmake_policy( SET CMP0048 NEW )
18
19 # Fetch GoogleTest and make build scripts available.
20 FetchContent_MakeAvailable( googletest )
```

- **That's not working either!** 

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- But how to set CMake command-line option `-Wno-dev` through `FetchContent`?
 - **That's not possible!** 
- OK, then let's set the CMake policy `CMP0048`:

```
15 ...
16 # Try to set policy CMP0048 for GoogleTest project.
17 cmake_policy( SET CMP0048 NEW )
18
19 # Fetch GoogleTest and make build scripts available.
20 FetchContent_MakeAvailable( googletest )
```

- **That's not working either!** 
- Because `cmake_minimum_required` is called in *GoogleTest's CMakeLists.txt* setting compatibility to CMake 2.6.2! 

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.  
11 ...  
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?
 - No, use `CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE`!

```
15 ...
16 # Require GoogleTest's top CMakeLists.txt to include a script
17 # before calling the project command which works around the problem.
18 set( CMAKE_PROJECT_gtest-distribution_INCLUDE_BEFORE
19     "${CMAKE_CURRENT_LIST_DIR}/GoogleTest-helper.cmake" )
20
21 # Fetch GoogleTest and make build scripts available.
22 FetchContent_MakeAvailable( gtest )
```

```
01 # .../GoogleTest-helper.cmake
02
03 cmake_policy( SET CMP0048 NEW )
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?
 - No, use `CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE`!

```
15 ...
16 # Require GoogleTest's top CMakeLists.txt to include a script
17 # before calling the project command which works around the problem.
18 set( CMAKE_PROJECT_gtest-distribution_INCLUDE_BEFORE
19     "${CMAKE_CURRENT_LIST_DIR}/GoogleTest-helper.cmake" )
20
21 # Fetch GoogleTest and make build scripts available.
22 FetchContent_MakeAvailable( gtest )
```

```
01 # .../GoogleTest-helper.cmake
02
03 cmake_policy( SET CMP0048 NEW )
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?
 - No, use `CMAKE_PROJECT <project-name> INCLUDE BEFORE!`

```
15 ...
16 # Require GoogleTest's top CMakeLists.txt to include a script
17 # before calling the project command which works around the problem.
18 set( CMAKE_PROJECT_gtest-distribution_INCLUDE_BEFORE
19     "${CMAKE_CURRENT_LIST_DIR}/GoogleTest-helper.cmake" )
20
21 # Fetch GoogleTest and make build scripts available.
22 FetchContent_MakeAvailable( gtest )
01 # .../GoogleTest-helper.cmake
02
03 cmake_policy( SET CMP0048 NEW )
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?
 - No, use `CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE`!

```
15 ...
16 # Require GoogleTest's top CMakeLists.txt to include a script
17 # before calling the project command which works around the problem.
18 set( CMAKE_PROJECT_gtest-distribution_INCLUDE_BEFORE
19     "${CMAKE_CURRENT_LIST_DIR}/GoogleTest-helper.cmake" )
20
21 # Fetch GoogleTest and make build scripts available.
22 FetchContent_MakeAvailable( gtest )
```

```
01 # .../GoogleTest-helper.cmake
02
03 cmake_policy( SET CMP0048 NEW )
```

- That's working! 😎

QUICK INTERLUDE

QUICK INTERLUDE

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt
02 # directly before calling `project` command
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script>
```

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt
02 # directly before calling `project` command
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script>
```

```
01 # Load script for each CMakeLists.txt
02 # directly after calling `project` command
03 set( CMAKE_PROJECT_INCLUDE <path-to-script>
```

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt  
02 # directly before calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for each CMakeLists.txt  
02 # directly after calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE <path-to-script> )
```

introduced in CMake 3.15

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt  
02 # directly before calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for each CMakeLists.txt  
02 # directly after calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for a specific CMakeLists.txt  
02 # directly before calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE_BEFORE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE <path-to-script> )
```

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt  
02 # directly before calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for each CMakeLists.txt  
02 # directly after calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for a specific CMakeLists.txt  
02 # directly before calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE_BEFORE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE <path-to-s
```

```
01 # Load script for a specific CMakeLists.txt  
02 # directly after calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE <path-to-s
```

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt  
02 # directly before calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for each CMakeLists.txt  
02 # directly after calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for a specific CMakeLists.txt  
02 # directly before calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE_BEFORE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE <path-to-script> )
```

will be introduced in CMake 3.17

```
01 # Load script for a specific CMakeLists.txt  
02 # directly after calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE <path-to-script> )
```

introduced in CMake 2.8.9

QUICK INTERLUDE

```
01 # Load script for each CMakeLists.txt  
02 # directly before calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE_BEFORE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for each CMakeLists.txt  
02 # directly after calling `project` command  
03 set( CMAKE_PROJECT_INCLUDE <path-to-script> )
```

introduced in CMake 3.15

```
01 # Load script for a specific CMakeLists.txt  
02 # directly before calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE_BEFORE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE <path-to-script> )
```

will be introduced in CMake 3.17

```
01 # Load script for a specific CMakeLists.txt  
02 # directly after calling `project` command  
03 # (but after CMAKE_PROJECT_INCLUDE).  
04 set( CMAKE_PROJECT_<project-name>_INCLUDE <path-to-script> )
```

introduced in CMake 2.8.9

REVISITING OUR PROBLEMS WITH *GOOGLETEST*

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?

- No, use `CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE`!

```
15 ...
16 # Require GoogleTest's top CMakeLists.txt to include a script
17 # before calling the project command which works around the problem.
18 set( CMAKE_PROJECT_gtest-distribution_INCLUDE_BEFORE
19     "${CMAKE_CURRENT_LIST_DIR}/GoogleTest-helper.cmake" )
20
21 # Fetch GoogleTest and make build scripts available.
22 FetchContent_MakeAvailable( gtest )
```

```
01 # .../GoogleTest-helper.cmake
02
03 cmake_policy( SET CMP0048 NEW )
```

That's working! 😎

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

Let's try to remove that annoying warning.

```
01 Policy CMP0048 is not set: project() command manages VERSION variables.
11 ...
12 This warning is for project developers. Use -Wno-dev to suppress it.
```

- Do I really have to patch *GoogleTest*'s `CMakeLists.txt` file?
 - No, use `CMAKE_PROJECT_<project-name>_INCLUDE_BEFORE`!

```
15 ...
16 # Require GoogleTest's top CMakeLists.txt to include a script
17 # before calling the project command which works around the problem.
18 set( CMAKE_PROJECT_gtest-distribution_INCLUDE_BEFORE
19     "${CMAKE_CURRENT_LIST_DIR}/GoogleTest-helper.cmake" )
20
21 # Fetch GoogleTest and make build scripts available.
22 FetchContent_MakeAvailable( gtest )
```

```
01 # .../GoogleTest-helper.cmake
02
03 cmake_policy( SET CMP0048 NEW )
```

~~That's working! 😎~~ That's only working with CMake 3.17 and newer! 😞

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

- Luckily, for *policy problems* there exists another solution:
 - Set a default-value of a policy, which will be used if it is unset.

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

- Luckily, for *policy problems* there exists another solution:
 - Set a default-value of a policy, which will be used if it is unset.
 - by setting: `CMAKE_POLICY_DEFAULT_CMP<NNNN>`

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

- Luckily, for *policy problems* there exists another solution:
 - Set a default-value of a policy, which will be used if it is unset.
 - by setting: `CMAKE_POLICY_DEFAULT_CMP<NNNN>`

```
15 ...
16 # Set default value for policy CMP0048 which will be used by
17 # GoogleTest's CMakeLists.txt scripts.
18 set( CMAKE_POLICY_DEFAULT_CMP0048 NEW )
19
20 # Fetch GoogleTest and make build scripts available.
21 FetchContent_MakeAvailable( googletest )
```

- That is working with all CMake versions! 😊

MORE PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

By the way:

- Beware of scripts loaded via `CMAKE_PROJECT_INCLUDE[_BEFORE]` when building external libraries.

MORE PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

By the way:

- Beware of scripts loaded via `CMAKE_PROJECT_INCLUDE[_BEFORE]` when building external libraries.

```
01  CMake Error at .../common-project-include-in:3 (include):  
02      include could not find load file:  
03  
04          .../project-meta-info.in
```

MORE PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

By the way:

- Beware of scripts loaded via `CMAKE_PROJECT_INCLUDE[_BEFORE]` when building external libraries.

```
01 CMake Error at .../common-project-include-in:3 (include):  
02   include could not find load file:  
03  
04     .../project-meta-info.in
```

- Need to unset variable temporarily.

```
19 ...
20 # Unset CMAKE_PROJECT_INCLUDE_BEFORE temporarily.
21 set( backup_CMAKE_PROJECT_INCLUDE_BEFORE ${CMAKE_PROJECT_INCLUDE_BEFORE} )
22 unset( CMAKE_PROJECT_INCLUDE_BEFORE )
23
24 # Fetch GoogleTest and make build scripts available.
25 FetchContent_MakeAvailable( googletest )
26
27 # Restore CMAKE_PROJECT_INCLUDE_BEFORE again.
28 set( CMAKE_PROJECT_INCLUDE_BEFORE ${backup_CMAKE_PROJECT_INCLUDE_BEFORE} )
29 unset( backup_CMAKE_PROJECT_INCLUDE_BEFORE )
```



source: <https://uip.me/wp-content/uploads/2013/03/one-more-thing.jpg>

MORE PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

MORE *GOOGLETTEST* PECULIARITIES

- *GoogleTest* targets do
 - **not use *namespace* syntax** and do
 - **not set *usage-requirements*.**

MORE PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

MORE *GoogleTest* PECULIARITIES

- *GoogleTest* targets do
 - **not use *namespace* syntax** and do
 - **not set *usage-requirements*.**

```
29 ...
30
31 # Create alias for targets.
32 if (NOT TARGET GTest::gtest)
33     add_library( GTest::gtest ALIAS gtest )
34 endif ()
35 if (NOT TARGET GTest::main)
36     add_library( GTest::main ALIAS gtest_main )
37 endif ()
38 if (NOT TARGET GMock::gmock)
39     target_link_libraries( gmock INTERFACE GTest::gtest )
40     add_library( GMock::gmock ALIAS gmock )
41 endif ()
42 if (NOT TARGET GMock::main)
43     target_link_libraries( gmock_main INTERFACE GTest::gtest )
44     add_library( GMock::main ALIAS gmock_main )
45 endif ()
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

- *GoogleTest* targets do
 - **not use *namespace* syntax** and do
 - **not set *usage-requirements*.**

```
29 ...
30
31 # Create alias for targets.
32 if (NOT TARGET GTest::gtest)
33     add_library( GTest::gtest ALIAS gtest )
34 endif ()
35 if (NOT TARGET GTest::main)
36     add_library( GTest::main ALIAS gtest_main )
37 endif ()
38 if (NOT TARGET GMock::gmock)
39     target_link_libraries( gmock INTERFACE GTest::gtest )
40     add_library( GMock::gmock ALIAS gmock )
41 endif ()
42 if (NOT TARGET GMock::main)
43     target_link_libraries( gmock_main INTERFACE GTest::gtest )
44     add_library( GMock::main ALIAS gmock_main )
45 endif ()
```

GOOGLETTEST

```
01 CMake Error at CMakeLists.txt:35 (target_link_libraries):
02 The plain signature for target_link_libraries has already been used with
03 the target "gmock". All uses of target_link_libraries with a target must
04 be either all-keyword or all-plain.
05
06 The uses of the plain signature are here:
07
08 * .../_deps/googletest-src/googletest/cmake/internal_utils.cmake:159 (target
09
10 CMake Error at CMakeLists.txt:40 (target_link_libraries):
11 The plain signature for target_link_libraries has already been used with
12 the target "gmock_main". All uses of target_link_libraries with a target
13 must be either all-keyword or all-plain.
14
15 The uses of the plain signature are here:
16
17 * .../_deps/googletest-src/googletest/cmake/internal_utils.cmake:159 (target
```

PROBLEMS WHEN BUILDING EXTERNAL LIBRARIES

GOOGLETTEST

- *GoogleTest* targets do
 - **not use *namespace* syntax** and do
 - **not set *usage-requirements*.**

```
29 ...
30
31 # Create alias for targets.
32 if (NOT TARGET GTest::gtest)
33     add_library( GTest::gtest ALIAS gtest )
34 endif ()
35 if (NOT TARGET GTest::main)
36     add_library( GTest::main ALIAS gtest_main )
37 endif ()
38 if (NOT TARGET GMock::gmock)
39     target_link_libraries( gmock GTest::gtest )      # Note: Cannot use INTERFACE here!
40     add_library( GMock::gmock ALIAS gmock )
41 endif ()
42 if (NOT TARGET GMock::main)
43     target_link_libraries( gmock_main GTest::gtest ) # Note: Cannot use INTERFACE here!
44     add_library( GMock::main ALIAS gmock_main )
45 endif ()
```

WHAT SHOULD YOU AT LEAST
TAKE HOME FROM THIS TALK?

TAKEAWAY

- Of course, use *Modern CMake!*

TAKEAWAY

- Of course, use *Modern CMake!*
- Use *newest CMake* version if possible. (Not older than CMake 3.15.)

TAKEAWAY

- Of course, use *Modern CMake!*
- Use *newest CMake* version if possible. (Not older than CMake 3.15.)
- Use `find_package` in **CONFIG mode** to search for pre-built external dependencies.

TAKEAWAY

- Of course, use *Modern CMake!*
- Use *newest CMake* version if possible. (Not older than CMake 3.15.)
- Use `find_package` in **CONFIG mode** to search for pre-built external dependencies.
- Use **FetchContent** to configure/build external dependencies with your project.

TAKEAWAY

- Of course, use *Modern CMake!*
 - Use *newest CMake* version if possible. (Not older than CMake 3.15.)
 - Use `find_package` in **CONFIG mode** to search for pre-built external dependencies.
 - Use **FetchContent** to configure/build external dependencies with your project.
 - Reduce boiler-plate and set local modifications by using
`CMAKE_PROJECT_INCLUDE[_BEFORE]` and
`CMAKE_PROJECT_<project-name>_INCLUDE[_BEFORE]`.
- Beware of interaction with **FetchContent**.

TAKEAWAY

- Of course, use *Modern CMake!*
- Use *newest CMake* version if possible. (Not older than CMake 3.15.)
- Use `find_package` in **CONFIG mode** to search for pre-built external dependencies.
- Use **FetchContent** to configure/build external dependencies with your project.
- Reduce boiler-plate and set local modifications by using
`CMAKE_PROJECT_INCLUDE[_BEFORE]` and
`CMAKE_PROJECT_<project-name>_INCLUDE[_BEFORE]`.
 - Beware of interaction with **FetchContent**.
- Use `find_package(Boost ...)` always with components!



That's all folks!

REFERENCES

- ***CMake's Reference-Documentation***
Read/Search at: <https://cmake.org/cmake/help/latest/index.html>
- ***Craig Scott's "Professional CMake: A Practical Guide" e-book***
Buy it at: <https://crascit.com/professional-cmake/>
- ***Craig Scott's "Deep CMake for Library Authors" talk***
Watch it at: <https://youtu.be/m0DwB4OvDXk>
- ***Deniz Bahadir's "More Modern CMake" talk***
Watch it at: <https://youtu.be/y7ndUhdQuU8>

I THINK YOU
SHOULD BE MORE
SPECIFIC HERE IN
STEP TWO

THANK YOU!
QUESTIONS?

