

MORE MODERN CMAKE

WORKING WITH CMAKE 3.12 AND LATER

Deniz Bahadir  **BENOCS**

cmake@deniz.bahadir.email

WHAT IS CMAKE?

WHAT IS CMAKE?

- CMake is a build-system *generator*

WHAT IS CMAKE?

- CMake is a build-system *generator*
 - **Not a build-system, though!**
 - generates *input files* for build-generators.
 - Supports: Make, Ninja, Visual Studio, XCode, ...

WHAT IS CMAKE?

- CMake is a build-system *generator*
 - **Not a build-system, though!**
 - generates *input files* for build-generators.
 - Supports: Make, Ninja, Visual Studio, XCode, ...
- It is cross-platform.
 - Supports *running* on: Linux, Windows, OSX, ...
 - Supports *building* cross-platform, too.
 - If the compiler supports that.

WHAT IS CMAKE?

- CMake is a build-system *generator*
 - **Not a build-system, though!**
 - generates *input files* for build-generators.
 - Supports: Make, Ninja, Visual Studio, XCode, ...
- It is cross-platform.
 - Supports *running* on: Linux, Windows, OSX, ...
 - Supports *building* cross-platform, too.
 - If the compiler supports that.
- Supports generating build-systems for multiple languages.
 - **C/C++ , FORTRAN, C#, CUDA...**

A (VERY) BRIEF HISTORY OF *CMAKE*.

A (VERY) BRIEF HISTORY OF *CMAKE*.

- CMake started around 1999/2000.
 - Not modern.

A (VERY) BRIEF HISTORY OF *CMAKE*.

- CMake started around 1999/2000.
 - Not modern.
- CMake is "modern" since version 3.0. (*around 2014*)
 - New concept: ***"everything is a (self-contained) target"***.

A (VERY) BRIEF HISTORY OF *CMAKE*.

- CMake started around 1999/2000.
 - Not modern.
- CMake is "modern" since version 3.0. (*around 2014*)
 - New concept: "***everything is a (self-contained) target***".
- CMake 3.11 was released (*March 2018*)
 - Unifies several commands.
- CMake 3.12 was released (*July 2018*)
 - Some of the big left-out tasks of *Modern CMake* were completed.

A (VERY) BRIEF HISTORY OF *CMAKE*.

- CMake started around 1999/2000.
 - Not modern.
- ⇒ **Traditional CMake**
- CMake is "modern" since version 3.0. (*around 2014*)
 - New concept: ***"everything is a (self-contained) target"***.
- ⇒ **Modern CMake**
- CMake 3.11 was released (*March 2018*)
 - Unifies several commands.
- CMake 3.12 was released (*July 2018*)
 - Some of the big left-out tasks of *Modern CMake* were completed.
- ⇒ **More Modern CMake**

SOME TERMS USED THROUGHOUT THIS TALK

BUILD-REQUIREMENTS OF A TARGET

BUILD-REQUIREMENTS OF A TARGET

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

BUILD-REQUIREMENTS OF A TARGET

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

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

USAGE-REQUIREMENTS OF A TARGET

USAGE-REQUIREMENTS OF A TARGET

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

USAGE-REQUIREMENTS OF A TARGET

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

"As a dependency of another target."

USAGE-REQUIREMENTS OF A TARGET

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

- source-files
 - *(but normally not)*
- include search-paths
- pre-processor macros
- link-dependencies
- compiler/linker-options
- compiler/linker-features
 - *(e.g. support for a C++-standard)*

TRADITIONAL CMAKE? MODERN CMAKE? WHAT'S THE DIFFERENCE?

COMPARISON

Traditional CMake

Modern CMake

COMPARISON

build-requirements are set on?

Traditional CMake

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

Modern CMake

on **targets***

* Or already on dependencies (as we will see later).

COMPARISON

	Traditional CMake	Modern CMake
<i>build-requirements</i> are set on?	on environment (mainly) e.g. <i>directory scope</i>	on targets *
keeping track of <i>usage-requirements</i>	via (cache-) variables	via targets (<i>keep track themselves</i>)

* Or already on dependencies (as we will see later).

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**

Modern CMake

on **targets***

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

automatic propagation

* Or already on dependencies (as we will see later).

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

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 (as we will see later).

** 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_link_libraries(     <target> PRIVATE <dependency>... )
09 target_link_options(       <target> PRIVATE <linker-option>... )
10 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_link_libraries(     <target> INTERFACE <dependency>... )
09 target_link_options(       <target> INTERFACE <linker-option>... )
10 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_link_libraries(     <target> PRIVATE <dependency>... )
09 target_link_options(       <target> PRIVATE <linker-option>... )
10 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_link_libraries(    <target> INTERFACE <dependency>... )
09 target_link_options(      <target> INTERFACE <linker-option>... )
10 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_link_libraries(     <target> PRIVATE <dependency>... )
09 target_link_options(       <target> PRIVATE <linker-option>... )
10 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_link_libraries(     <target> INTERFACE <dependency>... )
09 target_link_options(       <target> INTERFACE <linker-option>... )
10 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>... )
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_link_libraries(     <target> PUBLIC <dependency>... )
09 target_link_options(       <target> PUBLIC <linker-option>... )
10 target_link_directories(   <target> PUBLIC <linker-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_link_libraries(    <target> INTERFACE <dependency>... )
09 target_link_options(      <target> INTERFACE <linker-option>... )
10 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
07 # Adding build- and usage-requirements
08
09 target_include_directories( <target> PUBLIC <include-search-dir>... )
10 target_compile_definitions( <target> PUBLIC <macro-definitions>... )
11 target_compile_options(      <target> PUBLIC <compiler-option>... )
12 target_compile_features(     <target> PUBLIC <feature>... )
13 target_sources(              <target> PUBLIC <source-file>... )
14 target_link_libraries(       <target> PUBLIC <dependency>... )
15 target_link_options(         <target> PUBLIC <linker-option>... )
16 target_link_directories(    <target> PUBLIC <linker-search-dir>... )
17
18 target_compile_definitions( <target> INTERFACE <macro-definitions>... )
19 target_compile_options(      <target> INTERFACE <compiler-option>... )
20 target_compile_features(     <target> INTERFACE <feature>... )
21 target_sources(              <target> INTERFACE <source-file>... )
22 target_link_libraries(       <target> INTERFACE <dependency>... )
23 target_link_options(         <target> INTERFACE <linker-option>... )
24 target_link_directories(    <target> INTERFACE <linker-search-dir>... )
```

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

TRADITIONAL CMAKE VS. MODERN CMAKE VS. MORE MODERN CMAKE

DEMONSTRATED WITH EXAMPLE-PROJECT "CALCULATOR APP"

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
└── library/
    ├── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

TRADITIONAL CMAKE VS. MODERN CMAKE VS. MORE MODERN CMAKE

DEMONSTRATED WITH EXAMPLE-PROJECT "CALCULATOR APP"

- A *free* executable `FreeCalculator`
- A *non-free* executable `PremiumCalculator`
(with extended functionality)

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
└── library/
    ├── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

TRADITIONAL CMAKE VS. MODERN CMAKE VS. MORE MODERN CMAKE

DEMONSTRATED WITH EXAMPLE-PROJECT "CALCULATOR APP"

- A *free* executable `FreeCalculator`
 - dynamically linked with `Boost.Program_Options` shared-library.
- A *non-free* executable `PremiumCalculator` (with extended functionality)
 - dynamically linked with `Boost.Program_Options` shared-library.

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
└── library/
    ├── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

TRADITIONAL CMAKE VS. MODERN CMAKE VS. MORE MODERN CMAKE

DEMONSTRATED WITH EXAMPLE-PROJECT "CALCULATOR APP"

- A *free* executable `FreeCalculator`
 - dynamically linked with `Boost.Program_Options` shared-library.
 - dynamically linked with `basicmath` shared-library.
- A *non-free* executable `PremiumCalculator` (with extended functionality)
 - dynamically linked with `Boost.Program_Options` shared-library.
 - dynamically linked with `extmath` shared-library

```
.  └── CMakeLists.txt
  └── app/
    └── CMakeLists.txt
      └── src/
        └── key-file.cpp
        └── main.cpp
  └── external/
    └── CMakeLists.txt
      └── boost/
        └── CMakeLists.txt
          └── boost_outcome/
            └── CMakeLists.txt
  └── library/
    └── CMakeLists.txt
      └── include/
        └── MathAPI.h
        └── Math.h
      └── src/
        └── BasicMath.cpp
        └── ExtendedMath.cpp
        └── HeavyMath.cpp
```

TRADITIONAL CMAKE VS. MODERN CMAKE VS. MORE MODERN CMAKE

DEMONSTRATED WITH EXAMPLE-PROJECT "CALCULATOR APP"

- A *free* executable `FreeCalculator`
 - dynamically linked with `Boost.Program_Options` shared-library.
 - dynamically linked with `basicmath` shared-library.
- A *non-free* executable `PremiumCalculator` (with extended functionality)
 - dynamically linked with `Boost.Program_Options` shared-library.
 - dynamically linked with `extmath` shared-library, which itself is
 - dynamically linked with `Boost.Graph` shared-library.

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    └── boost/
        └── CMakeLists.txt
        └── boost_outcome/
            └── CMakeLists.txt
└── library/
    ├── CMakeLists.txt
    └── include/
        ├── MathAPI.h
        └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

TRADITIONAL CMAKE VS. MODERN CMAKE VS. MORE MODERN CMAKE

DEMONSTRATED WITH EXAMPLE-PROJECT "CALCULATOR APP"

- A *free* executable `FreeCalculator`
 - dynamically linked with `Boost.Program_Options` shared-library.
 - dynamically linked with `basicmath` shared-library.
- A *non-free* executable `PremiumCalculator` (with extended functionality)
 - dynamically linked with `Boost.Program_Options` shared-library.
 - dynamically linked with `extmath` shared-library, which itself is
 - dynamically linked with `Boost.Graph` shared-library.
- All executables and libraries use `Boost.Outcome` header-only library internally.

```
.  └── CMakeLists.txt
    └── app/
        └── CMakeLists.txt
            └── src/
                └── key-file.cpp
                    └── main.cpp
    └── external/
        └── CMakeLists.txt
            └── boost/
                └── CMakeLists.txt
                    └── boost_outcome/
                        └── CMakeLists.txt
    └── library/
        └── CMakeLists.txt
            └── include/
                └── MathAPI.h
            └── src/
                └── BasicMath.cpp
                    └── ExtendedMath.cpp
                        └── HeavyMath.cpp
```

FINDING EXTERNAL DEPENDENCY

Boost

FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```

- Using `find_package` to locate *Boost*.

```
.
  CMakeLists.txt
  app/
    CMakeLists.txt
    src/
      key-file.cpp
      main.cpp
  external/
    CMakeLists.txt
    boost/
      CMakeLists.txt
      boost_outcome/
        CMakeLists.txt
  library/
    CMakeLists.txt
    include/
      MathAPI.h
      Math.h
    src/
      BasicMath.cpp
      ExtendedMath.cpp
      HeavyMath.cpp
```

FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```

- Using `find_package` to locate *Boost*.

```
.
  CMakeLists.txt
  app/
    CMakeLists.txt
    src/
      key-file.cpp
      main.cpp
  external/
    CMakeLists.txt
    boost/
      CMakeLists.txt
    boost_outcome/
      CMakeLists.txt
  library/
    CMakeLists.txt
    include/
      MathAPI.h
      Math.h
    src/
      BasicMath.cpp
      ExtendedMath.cpp
      HeavyMath.cpp
```

FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```

- Using `find_package` to locate *Boost*.
- If found, sets variables:
 - `Boost_INCLUDE_DIRS`, containing include-path
 - `Boost_LIBRARIES`, containing file-paths to shared libraries

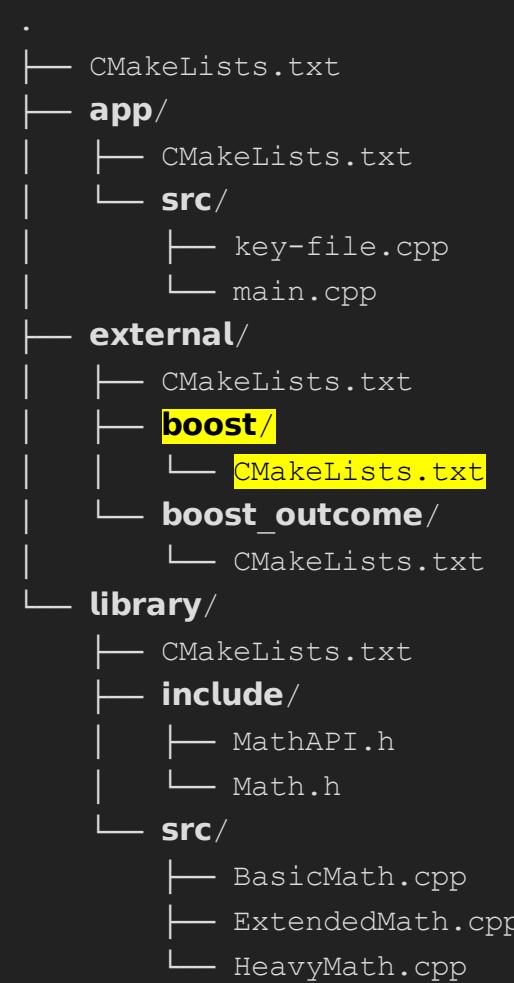
```
.
  CMakeLists.txt
  app/
    CMakeLists.txt
    src/
      key-file.cpp
      main.cpp
  external/
    CMakeLists.txt
    boost/
      CMakeLists.txt
      boost_outcome/
        CMakeLists.txt
  library/
    CMakeLists.txt
    include/
      MathAPI.h
      Math.h
    src/
      BasicMath.cpp
      ExtendedMath.cpp
      HeavyMath.cpp
```

FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```

- Using `find_package` to locate *Boost*.
- If found, sets variables:
 - `Boost_INCLUDE_DIRS`, containing include-path
 - `Boost_LIBRARIES`, containing file-paths to shared libraries
- *Modern CMake* additionally creates `IMPORTED` targets:
 - `Boost::boost`
 - `Boost::program_options`
 - `Boost::graph`

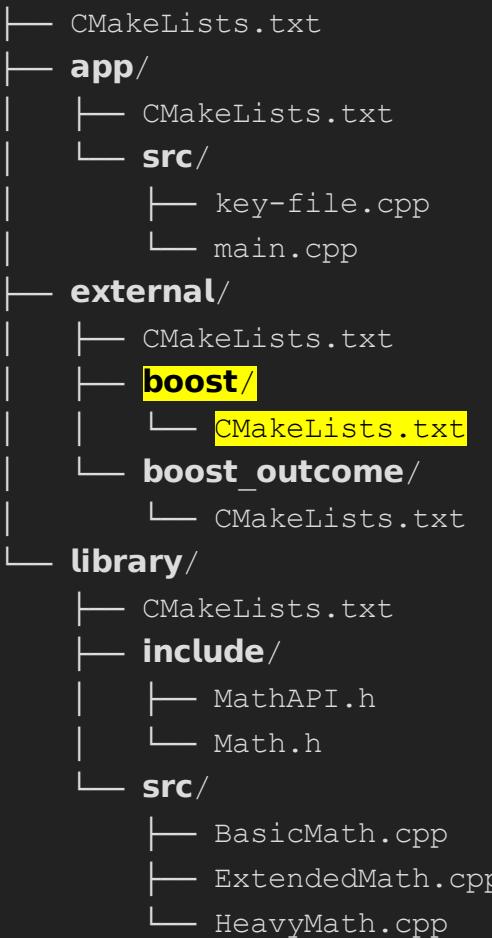


FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```

- Using `find_package` to locate *Boost*.
- If found, sets variables:
 - `Boost_INCLUDE_DIRS`, containing include-path
 - `Boost_LIBRARIES`, containing file-paths to shared libraries
- *Modern CMake* additionally creates `IMPORTED` targets:
 - `Boost::boost`
 - `Boost::program_options`
 - `Boost::graph`
- `IMPORTED` targets carry and can propagate their *usage-requirements*.



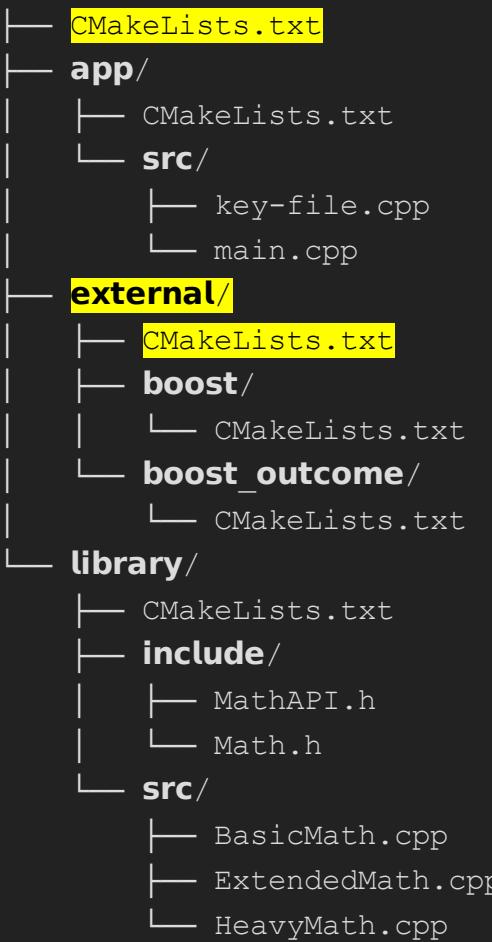
FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

- Variables and `IMPORTED` targets created by `find_package` have **non-global scope** (by convention).

```
01 #./CMakeLists.txt
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( Example_for_CMake )
06 set( VERSION 2.8.10 )
07 set( DESCRIPTION "Example project for CMake" )
08
09 # Always use '-fPIC'/'-fPIE' option.
10 set( CMAKE_POSITION_INDEPENDENT_CODE ON )
11
12 # Make external libraries globally available.
13 add_subdirectory( external ) # Does NOT work for Boost!
14
15
16 # Create targets for building the (local) libraries.
17 add_subdirectory( library )
18
19 # Create the targets for the entire example-app.
20 add_subdirectory( app )
```

```
01 # ./external/CMakeLists.txt
02
03 add_subdirectory( boost )
04 add_subdirectory( boost_outcome )
```



FINDING EXTERNAL DEPENDENCY - *Boost*

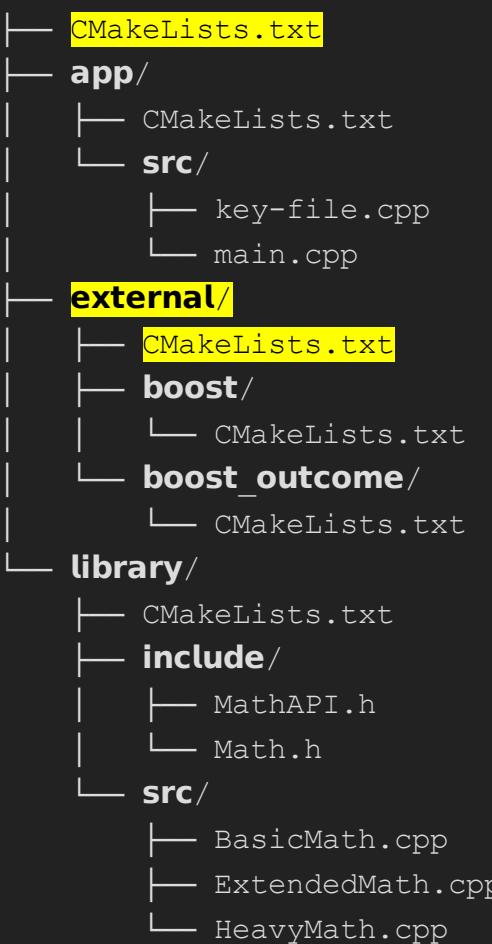
TRADITIONAL AND MODERN CMAKE WAY

- Variables and `IMPORTED` targets created by `find_package` have **non-global scope** (by convention).

```
01 #./CMakeLists.txt
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( Example_for_CMake )
06 set( VERSION 2.8.10 )
07 set( DESCRIPTION "Example project for CMake" )
08
09 # Always use '-fPIC'/'-fPIE' option.
10 set( CMAKE_POSITION_INDEPENDENT_CODE ON )
11
12 # Make external libraries globally available.
13 add_subdirectory( external ) # Does NOT work for Boost!
14
15
16 # Create targets for building the (local) libraries.
17 add_subdirectory( library )
18
19 # Create the targets for the entire example-app.
20 add_subdirectory( app )
```

```
01 # ./external/CMakeLists.txt
02
03 add_subdirectory( boost )
04 add_subdirectory( boost_outcome )
```

- Using `add_subdirectory` is preferred but does not help here.



FINDING EXTERNAL DEPENDENCY - *Boost*

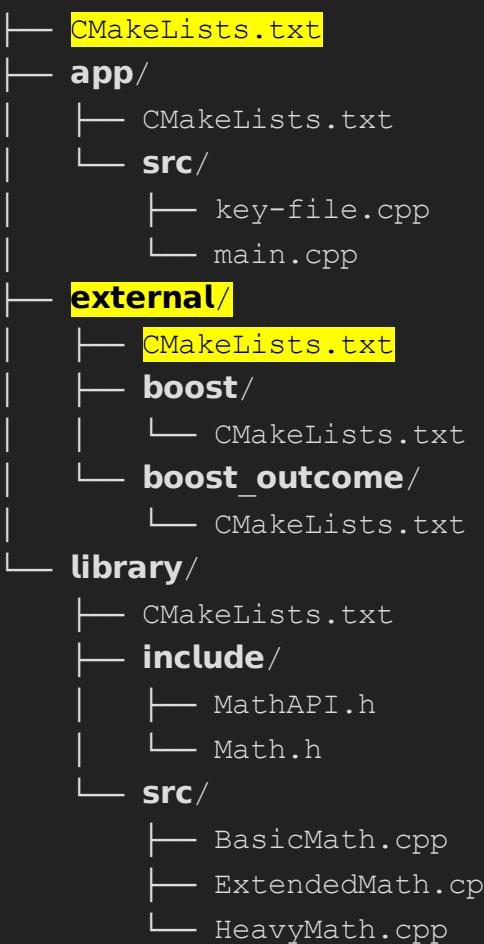
TRADITIONAL AND MODERN CMAKE WAY

- Variables and `IMPORTED` targets created by `find_package` have **non-global scope** (by convention).

```
01 #./CMakeLists.txt -- Traditional/Modern CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( Example_for_CMake )
06 set( VERSION 2.8.10 )
07 set( DESCRIPTION "Example project for CMake" )
08
09 # Always use '-fPIC'/'-fPIE' option.
10 set( CMAKE_POSITION_INDEPENDENT_CODE ON )
11
12 # Make external libraries globally available.
13 add_subdirectory( external )
14 include( external/boost/CMakeLists.txt )
15
16 # Create targets for building the (local) libraries.
17 add_subdirectory( library )
18
19 # Create the targets for the entire example-app.
20 add_subdirectory( app )
```

```
01 # ./external/CMakeLists.txt -- Traditional/Modern CMake
02
03
04 add_subdirectory( boost_outcome )
```

- Using `add_subdirectory` is preferred but does not help here.
- `CMakeLists.txt` file for *Boost* needs to be `included` into the top `CMakeLists.txt` file directly.



FINDING EXTERNAL DEPENDENCY - *Boost*

TRADITIONAL AND MODERN CMAKE WAY

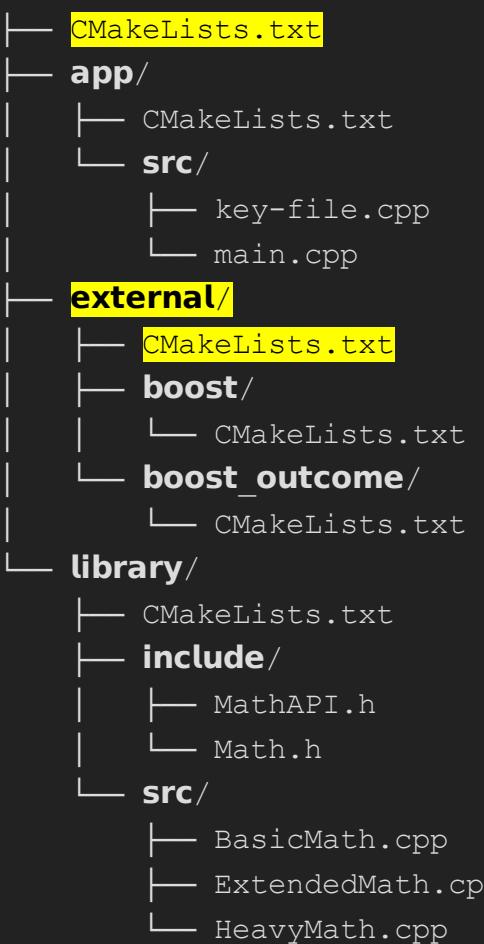
- Variables and `IMPORTED` targets created by `find_package` have **non-global scope** (by convention).

```
01 ./CMakeLists.txt -- Traditional/Modern CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( Example_for_CMake )
06 set( VERSION 2.8.10 )
07 set( DESCRIPTION "Example project for CMake" )
08
09 # Always use '-fPIC'/'-fPIE' option.
10 set( CMAKE_POSITION_INDEPENDENT_CODE ON )
11
12 # Make external libraries globally available.
13 add_subdirectory( external )
14 include( external/boost/CMakeLists.txt )
15
16 # Create targets for building the (local) libraries.
17 add_subdirectory( library )
18
19 # Create the targets for the entire example-app.
20 add_subdirectory( app )
```

```
01 # ./external/CMakeLists.txt -- Traditional/Modern CMake
02
03
04 add_subdirectory( boost_outcome )
```

- Using `add_subdirectory` is preferred but does not help here.
- `CMakeLists.txt` file for *Boost* needs to be `included` into the top `CMakeLists.txt` file directly.

HOWEVER...



FINDING EXTERNAL DEPENDENCY - *Boost*

MORE MODERN CMAKE WAY

- Since CMake 3.11 `IMPORTED` targets can be promoted to **global scope**
 - by setting the `IMPORTED_GLOBAL` property to `TRUE`.

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```



```
01 # ./external/boost/CMakeLists.txt -- More Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
16
17 # Make found targets globally available.
18 if ( Boost_FOUND )
19   set_target_properties( Boost::boost
20                         Boost::program_options
21                         Boost::graph
22                         PROPERTIES IMPORTED_GLOBAL TRUE )
23 endif ()
```

```
.   CMakeLists.txt
  +-- app/
  |   +-- CMakeLists.txt
  |   +-- src/
  |       +-- key-file.cpp
  |       +-- main.cpp
  +-- external
  |   +-- CMakeLists.txt
  |   +-- boost/
  |       +-- CMakeLists.txt
  |   +-- boost_outcome/
  |       +-- CMakeLists.txt
  +-- library/
  |   +-- CMakeLists.txt
  |   +-- include/
  |       +-- MathAPI.h
  |       +-- Math.h
  +-- src/
      +-- BasicMath.cpp
      +-- ExtendedMath.cpp
      +-- HeavyMath.cpp
```

FINDING EXTERNAL DEPENDENCY - *Boost*

MORE MODERN CMAKE WAY

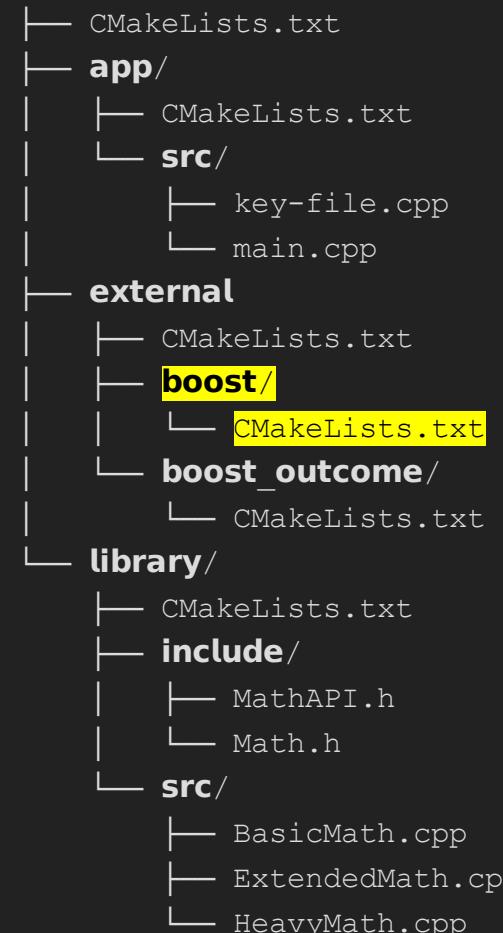
- Since CMake 3.11 `IMPORTED` targets can be promoted to **global scope**
 - by setting the `IMPORTED_GLOBAL` property to `TRUE`.

```
01 # ./external/boost/CMakeLists.txt -- Traditional/Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
```



```
01 # ./external/boost/CMakeLists.txt -- More Modern CMake
02
03 set( BOOST_VERSION 1.58.0 )
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_ADDITIONAL_VERSIONS "${BOOST_VERSION}" )
10 set( Boost_COMPILER             "-gcc" )
11
12 # Search for Boost libraries.
13 find_package( Boost ${BOOST_VERSION} EXACT REQUIRED
14   COMPONENTS program_options
15   graph )
16
17 # Make found targets globally available.
18 if ( Boost_FOUND )
19   set_target_properties( Boost::boost
20                         Boost::program_options
21                         Boost::graph
22                         PROPERTIES IMPORTED_GLOBAL TRUE )
23 endif ()
```

- Promoted/Global `IMPORTED` targets *cannot* be demoted!



FINDING EXTERNAL DEPENDENCY - *Boost*

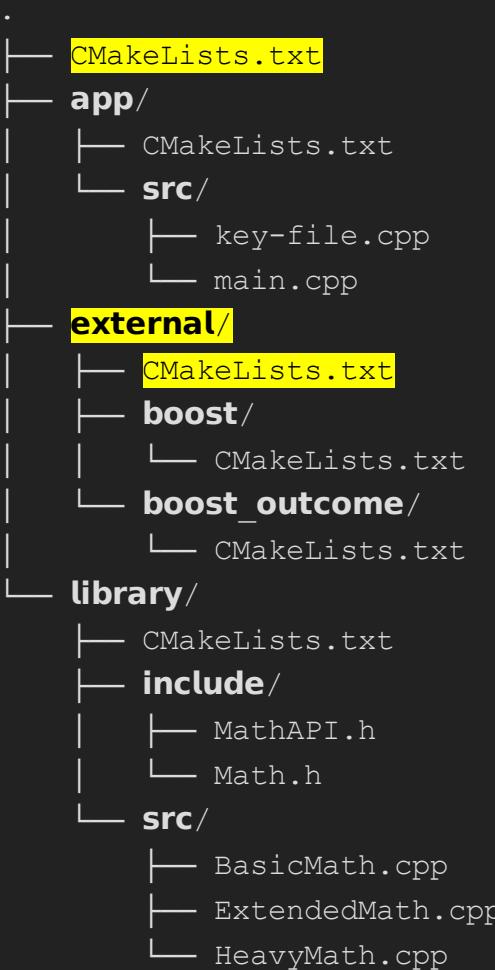
MORE MODERN CMAKE WAY

- With **IMPORTED** targets promoted to **global** scope, this works as desired.

```
01 #./CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( Example_for_CMake
06     VERSION 3.11
07     DESCRIPTION "Example project for CMake" )
08
09 # Always use '-fPIC'/'-fPIE' option.
10 set( CMAKE_POSITION_INDEPENDENT_CODE ON )
11
12 # Make external libraries globally available.
13 add_subdirectory( external ) # Does also work for Boost!
14
15 # Create targets for building the (local) libraries.
16 add_subdirectory( library )
17
18 # Create the targets for the entire example-app.
19 add_subdirectory( app )
```

```
01 # ./external/CMakeLists.txt -- More Modern CMake
02
03 add_subdirectory( boost )
04 add_subdirectory( boost_outcome )
```

- Using **add_subdirectory** is preferred and now works here.



FINDING EXTERNAL DEPENDENCY - *Boost*

MORE MODERN CMAKE WAY

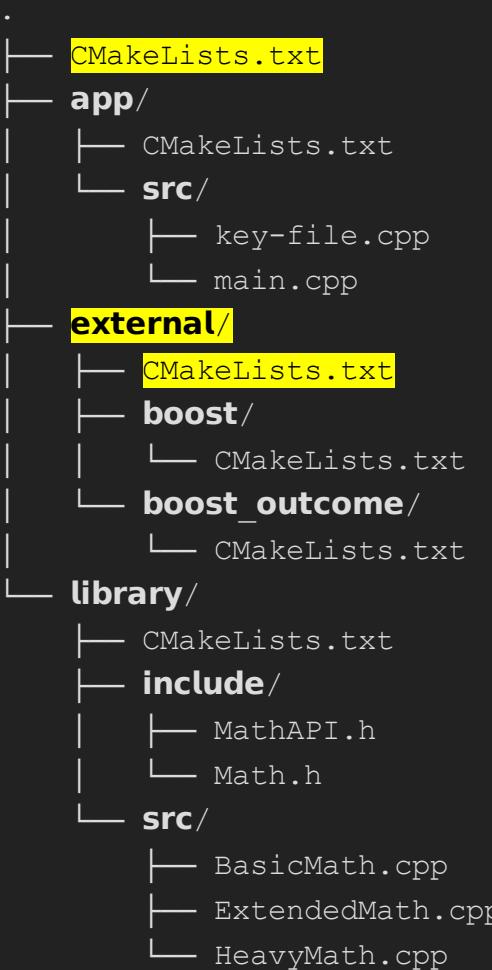
- With `IMPORTED` targets promoted to **global** scope, this works as desired.

```
01 #./CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( Example_for_CMake
06     VERSION 3.11
07     DESCRIPTION "Example project for CMake" )
08
09 # Always use '-fPIC'/'-fPIE' option.
10 set( CMAKE_POSITION_INDEPENDENT_CODE ON )
11
12 # Make external libraries globally available.
13 add_subdirectory( external ) # Does also work for Boost!
14
15 # Create targets for building the (local) libraries.
16 add_subdirectory( library )
17
18 # Create the targets for the entire example-app.
19 add_subdirectory( app )
```

```
01 # ./external/CMakeLists.txt -- More Modern CMake
02
03 add_subdirectory( boost )
04 add_subdirectory( boost_outcome )
```

- Using `add_subdirectory` is preferred and now works here.

- You need to use the `IMPORTED` targets for *Boost*.
- The variables created by `find_package` for *Boost* are not promoted to global scope!



CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY

BOOST.OUTCOME

CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY - *Boost.Outcome*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost_outcome/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( External.Reviewed_Boost.outcome )
06 set( VERSION 2.0 )
07 set( DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome UNKNOWN
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       MY_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       MY_COMPILE_FEATURES "-std=c++14" )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    └── boost/
        └── CMakeLists.txt
            └── boost_outcome/
                └── CMakeLists.txt
└── library/
    ├── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

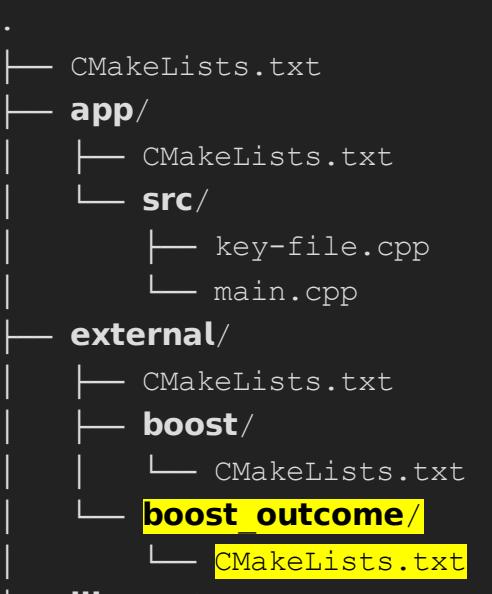
CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY - *Boost.Outcome*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost_outcome/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( External.Reviewed_Boost.outcome )
06 set( VERSION 2.0 )
07 set( DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome UNKNOWN
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       MY_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       MY_COMPILE_FEATURES "-std=c++14" )
```



```
01 # ./external/boost_outcome/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( External.Reviewed_Boost.outcome
06             VERSION 2.0
07             DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome INTERFACE
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       INTERFACE_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       INTERFACE_COMPILE_FEATURES cxx_std_14 )
```



- For *usage-requirements* the `INTERFACE_...` properties need to be set.

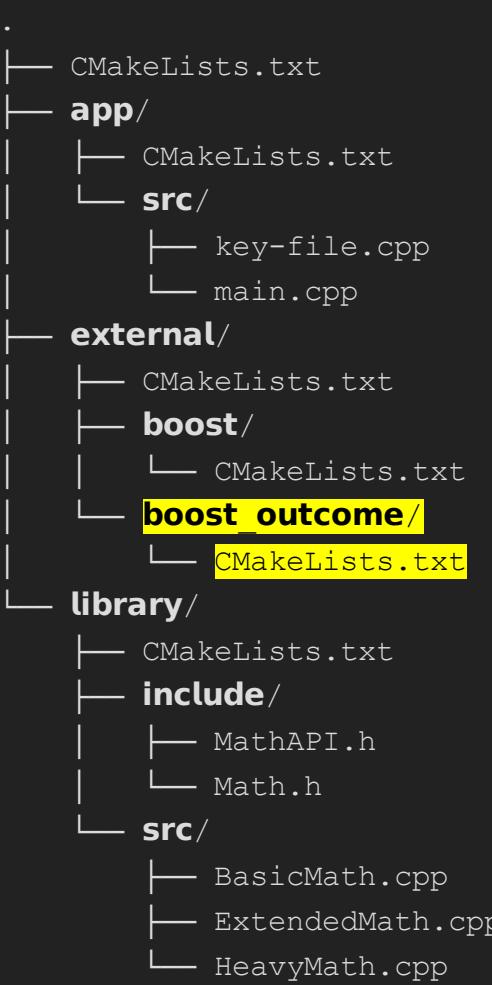
CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY - *Boost.Outcome*

TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost_outcome/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( External.Reviewed_Boost.outcome )
06 set( VERSION 2.0 )
07 set( DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome UNKNOWN
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       MY_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       MY_COMPILE_FEATURES "-std=c++14" )
```



```
01 # ./external/boost_outcome/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( External.Reviewed_Boost.outcome
06           VERSION 2.0
07           DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome INTERFACE
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       INTERFACE_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       INTERFACE_COMPILE_FEATURES cxx_std_14 )
```



- For *usage-requirements* the `INTERFACE_...` properties need to be set.
- Sadly, using `target_...` commands does not work with `IMPORTED` libraries.

CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY - *Boost.Outcome*

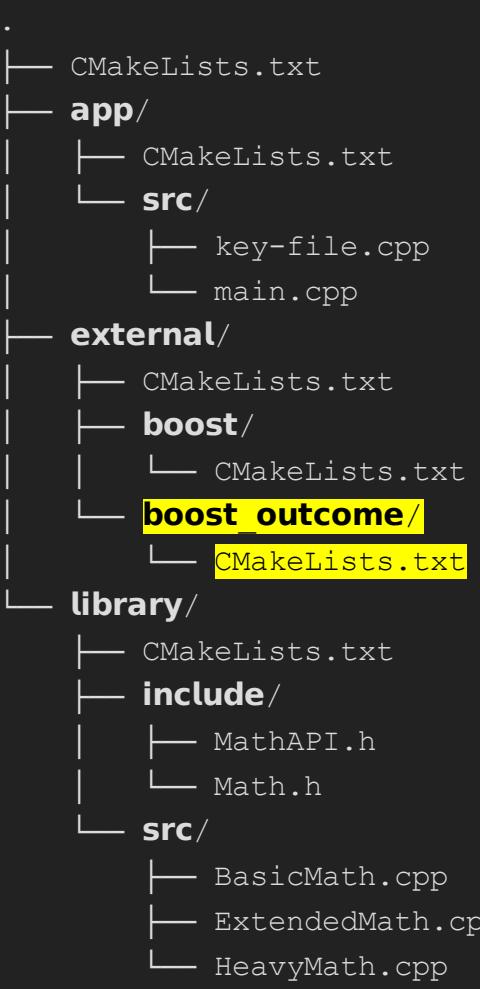
TRADITIONAL AND MODERN CMAKE WAY

```
01 # ./external/boost_outcome/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( External.Reviewed_Boost.outcome )
06 set( VERSION 2.0 )
07 set( DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome UNKNOWN
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                         MY_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                         MY_COMPILE_FEATURES "-std=c++14" )
```



```
01 # ./external/boost_outcome/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( External.Reviewed_Boost.outcome
06             VERSION 2.0
07             DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome INTERFACE
11             IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                         INTERFACE_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                         INTERFACE_COMPILE_FEATURES cxx_std_14 )
```

- For *usage-requirements* the `INTERFACE_...` properties need to be set.
- Sadly, using `target_...` commands does not work with `IMPORTED` libraries.



HOWEVER...

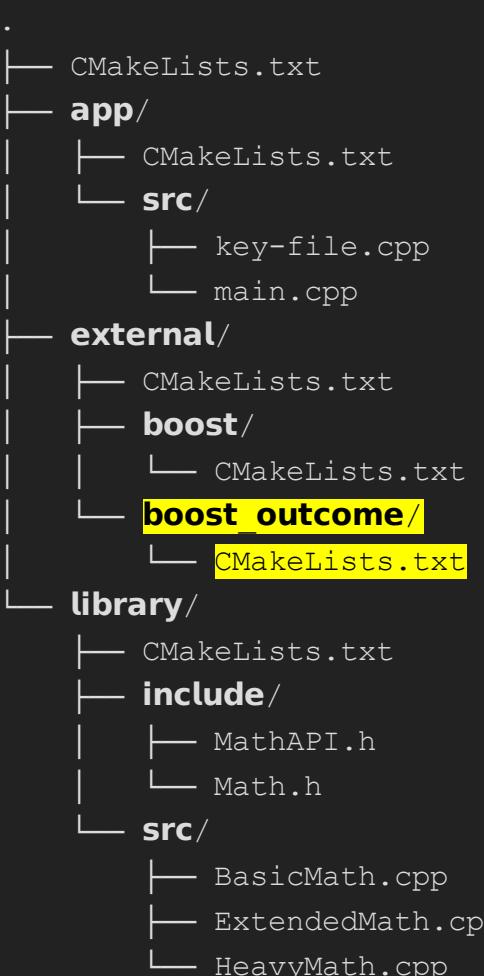
CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY - *Boost.Outcome*

MORE MODERN CMAKE WAY

- Since CMake 3.11 `target_...` commands can be used to set *usage-requirements* of `IMPORTED` targets (as for all other targets).

```
01 # ./external/boost_outcome/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( External.Reviewed_Boost.outcome
06           VERSION 2.0
07           DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome INTERFACE
11               IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       INTERFACE_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       INTERFACE_COMPILE_FEATURES cxx_std_14 )
```

```
01 # ./external/boost_outcome/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( External.Reviewed_Boost.outcome
06           VERSION 2.0
07           DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome INTERFACE
11               IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 target_include_directories( Boost::outcome SYSTEM
15                             INTERFACE /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 target_compile_features( Boost::outcome
18                         INTERFACE cxx_std_14 )
```



CREATING IMPORTED TARGET FOR EXTERNAL DEPENDENCY - *Boost.Outcome*

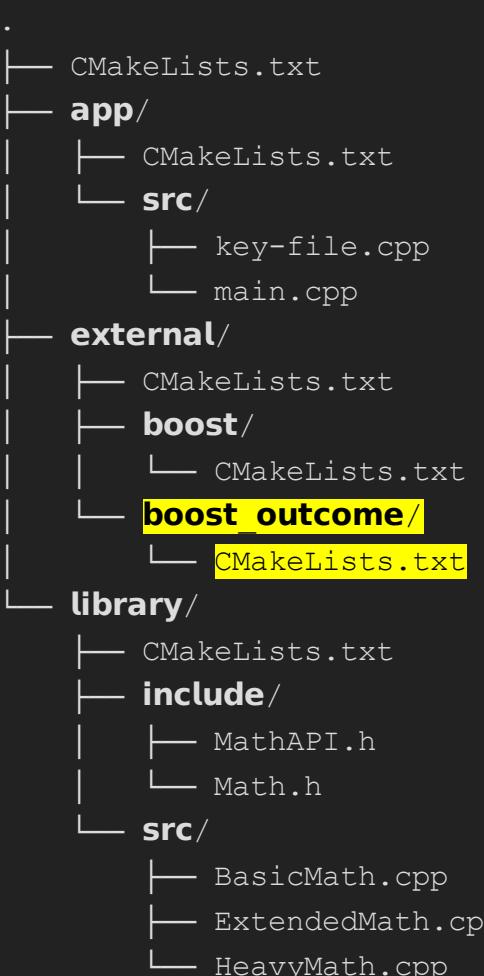
MORE MODERN CMAKE WAY

- Since CMake 3.11 `target_...` commands can be used to set *usage-requirements* of `IMPORTED` targets (as for all other targets).

```
01 # ./external/boost_outcome/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( External.Reviewed_Boost.outcome
06           VERSION 2.0
07           DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Boost::outcome INTERFACE
11               IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 set_target_properties( Boost::outcome PROPERTIES
15                       INTERFACE_INCLUDE_DIRS /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 set_target_properties( Boost::outcome PROPERTIES
18                       INTERFACE_COMPILE_FEATURES cxx_std_14 )
```

```
01 # ./external/boost_outcome/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( External.Reviewed_Boost.outcome
06           VERSION 2.0
07           DESCRIPTION "Boost.Outcome header-only lib." )
08
09 # Provide target for "Boost.Outcome" library.
10 add_library( Reviewed_Boost.outcome INTERFACE
11               IMPORTED GLOBAL )
12 # Store include search-path containing headers
13 # of "Boost.Outcome".
14 target_include_directories( Reviewed_Boost.outcome SYSTEM
15                           INTERFACE /opt/boost-outcome/include )
16 # Require at least compiling with C++14.
17 target_compile_features( Reviewed_Boost.outcome
18                         INTERFACE cxx_std_14 )
19
20 # Create an alias for "Boost.Outcome".
21 add_library( Boost::outcome ALIAS Reviewed_Boost.outcome )
```

- Even `ALIAS`ing `IMPORTED` targets is now possible.



CREATING LIBRARIES

BASICMATH / EXTMATH

CREATING LIBRARIES – AN OVERVIEW

TRADITIONAL CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
```

```
29
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36
37 # A shared library for advanced-math functionality.
38 add_library( extmath SHARED
39     "src/ExtendedMath.cpp" # Premium-content!
40     $<TARGET_OBJECTS:basicmath_ObjLib> )
41 # The usage-requirements of the OBJECT-library (and its
42 # direct dependency) are already set in directory scope
43 # (above in lines 19 to 28).
44
45 # Target 'extmath' requires include-path and needs
46 # to link to library-file of its extra dependency
47 # "Boost.Graph".
48 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
49 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
library/
    └── CMakeLists.txt
include/
    ├── MathAPI.h
    └── Math.h
src/
    ├── BasicMath.cpp
    ├── ExtendedMath.cpp
    └── HeavyMath.cpp
```

CREATING LIBRARIES – AN OVERVIEW

TRADITIONAL CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
```

```
29
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36
37 # A shared library for advanced-math functionality.
38 add_library( extmath SHARED
39     "src/ExtendedMath.cpp" # Premium-content!
40     $<TARGET_OBJECTS:basicmath_ObjLib> )
41 # The usage-requirements of the OBJECT-library (and its
42 # direct dependency) are already set in directory scope
43 # (above in lines 19 to 28).
44
45 # Target 'extmath' requires include-path and needs
46 # to link to library-file of its extra dependency
47 # "Boost.Graph".
48 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
49 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
library/
    └── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

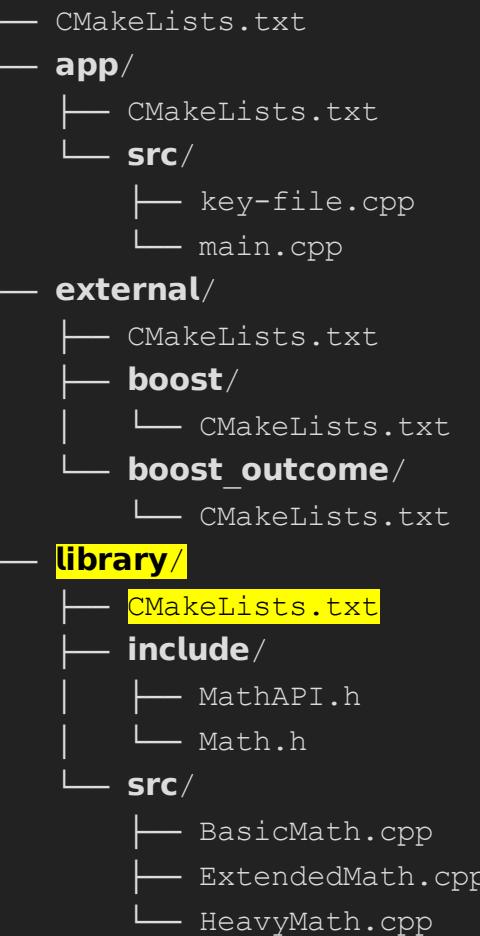
OVERWHELMING?

LET'S LOOK AT IT STEP BY STEP...

CREATING TARGETS AND ADDING SOURCES

TRADITIONAL AND MODERN CMAKE WAY

- First, create a common **OBJECT** library for both libraries
 - so that long-compiling files only need to be compiled once!

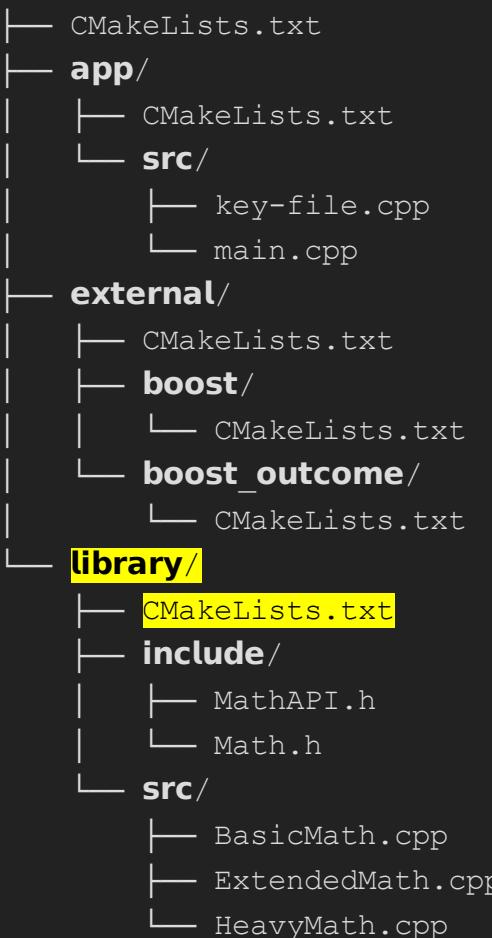


```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17 ...
```

CREATING TARGETS AND ADDING SOURCES

TRADITIONAL AND *Modern CMake* WAY

- First, create a common `OBJECT` library for both libraries
 - so that long-compiling files only need to be compiled once!
- *Modern CMake* allows to add intention to the source-files, by using `target_sources` command.



```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17 ...
```



```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06           VERSION 1.0.0
07           DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13                 PRIVATE    "src/BasicMath.cpp"
14                 PUBLIC     "src/HeavyMath.cpp" # Takes loooooong to compile!
15                 INTERFACE  "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16                 INTERFACE  "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```

CREATING TARGETS AND ADDING SOURCES

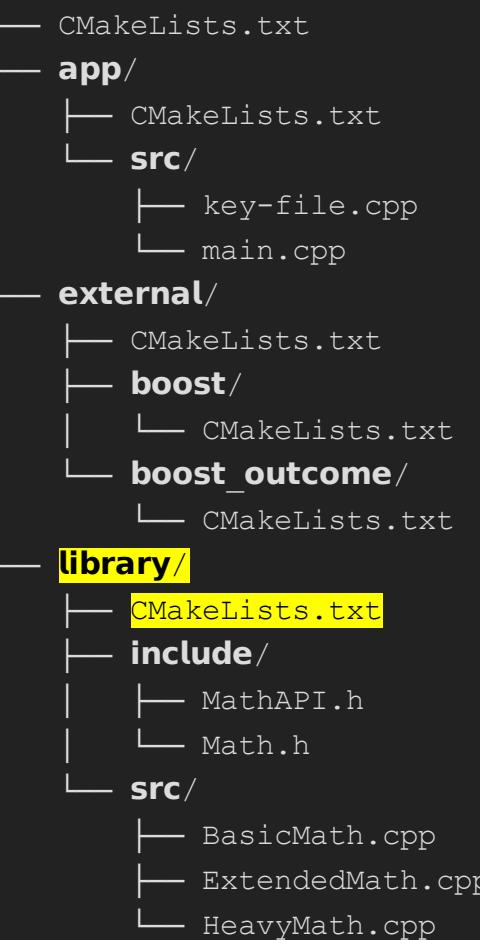
TRADITIONAL AND *Modern CMake* WAY

- First, create a common `OBJECT` library for both libraries
 - so that long-compiling files only need to be compiled once!
- *Modern CMake* allows to add intention to the source-files, by using `target_sources` command.
- Sadly, `add_library` requires at least one source-file argument.

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17 ...
```



```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06           VERSION 1.0.0
07           DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13                 PRIVATE    "src/BasicMath.cpp"
14                 "src/HeavyMath.cpp" # Takes loooooong to compile!
15                 PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16                 INTERFACE  "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```



CREATING TARGETS AND ADDING SOURCES

TRADITIONAL AND *Modern CMake* WAY

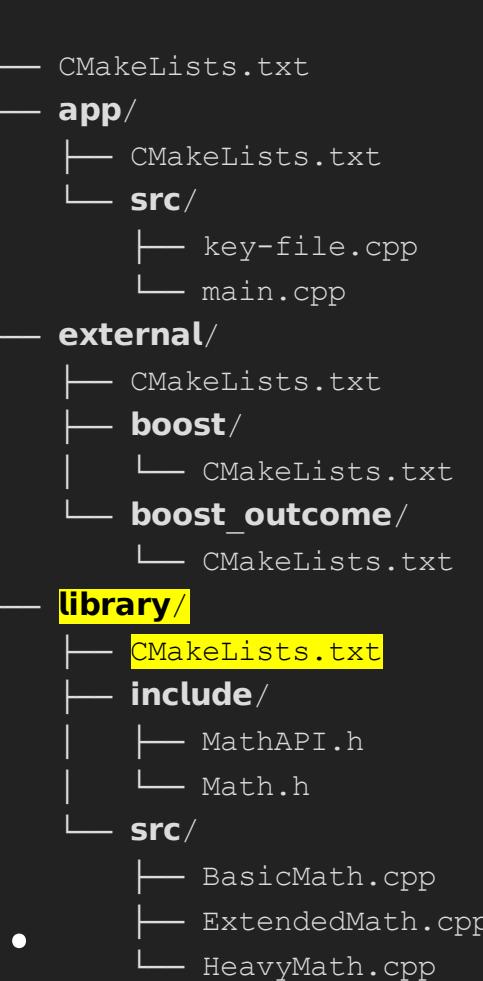
- First, create a common `OBJECT` library for both libraries
 - so that long-compiling files only need to be compiled once!
- *Modern CMake* allows to add intention to the source-files, by using `target_sources` command.
- Sadly, `add_library` requires at least one source-file argument.

HOWEVER...

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17 ...
```



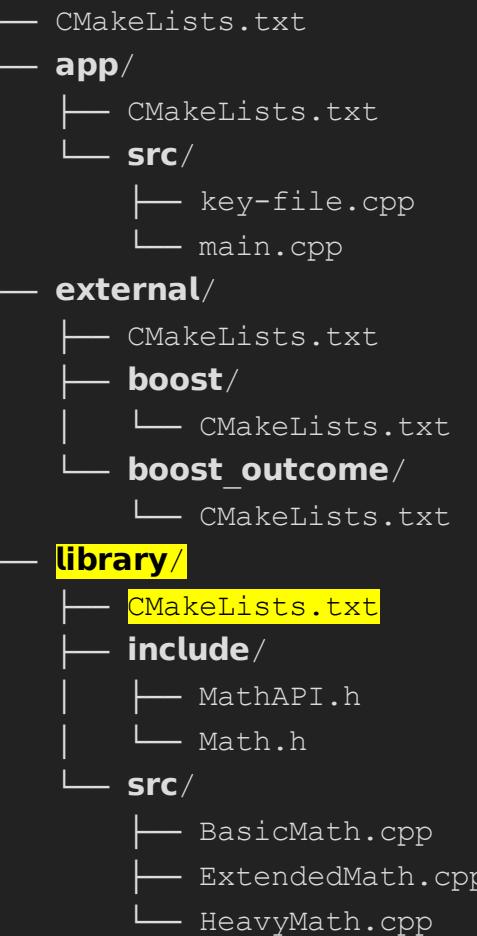
```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06           VERSION 1.0.0
07           DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13                 PRIVATE    "src/BasicMath.cpp"
14                 "src/HeavyMath.cpp" # Takes loooooong to compile!
15                 PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16                 INTERFACE  "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```



CREATING TARGETS AND ADDING SOURCES

MORE MODERN CMAKE WAY

- Since CMake 3.11 the `add_library` command can be used without any source-parameters.



```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06             VERSION 1.0.0
07             DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13                 PRIVATE    "src/BasicMath.cpp"
14                 "src/HeavyMath.cpp" # Takes loooooong to compile!
15                 PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16                 INTERFACE   "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```



```
01 # ./library/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( MathLibs
06             VERSION 1.0.0
07             DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT )
12 target_sources( basicmath_ObjLib
13                 PRIVATE    "src/BasicMath.cpp"
14                 "src/HeavyMath.cpp" # Takes loooooong to compile!
15                 PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16                 INTERFACE   "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```

CREATING TARGETS AND ADDING SOURCES

MORE MODERN CMAKE WAY

- Since CMake 3.11 the `add_library` command can be used without any source-parameters.
 - Of course, sources need to be added later via `target_sources` command, or CMake will complain.

```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13     PRIVATE    "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE   "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```



```
01 # ./library/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT )
12 target_sources( basicmath_ObjLib
13     PRIVATE    "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE   "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```

```
.   CMakeLists.txt
  + app/
  |   CMakeLists.txt
  |   + src/
  |   |   key-file.cpp
  |   |   main.cpp
  + external/
  |   CMakeLists.txt
  + boost/
  |   CMakeLists.txt
  + boost_outcome/
  |   CMakeLists.txt
  + library/
  |   CMakeLists.txt
  |   + include/
  |   |   MathAPI.h
  |   |   Math.h
  |   + src/
  |   |   BasicMath.cpp
  |   |   ExtendedMath.cpp
  |   |   HeavyMath.cpp
```

CREATING TARGETS AND ADDING SOURCES

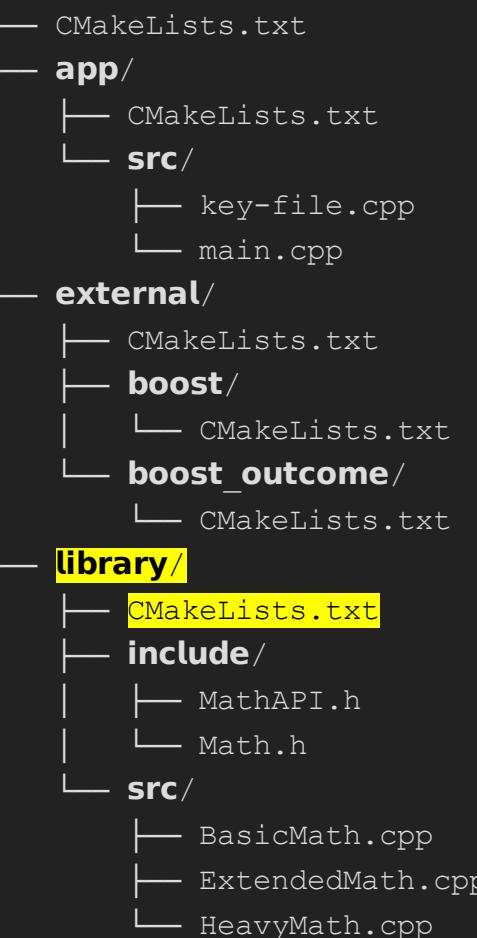
MORE MODERN CMAKE WAY

- Since CMake 3.11 the `add_library` command can be used without any source-parameters.
 - Of course, sources need to be added later via `target_sources` command, or CMake will complain.
- The same applies to `add_executable` command.

```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13     PRIVATE    "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE   "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```



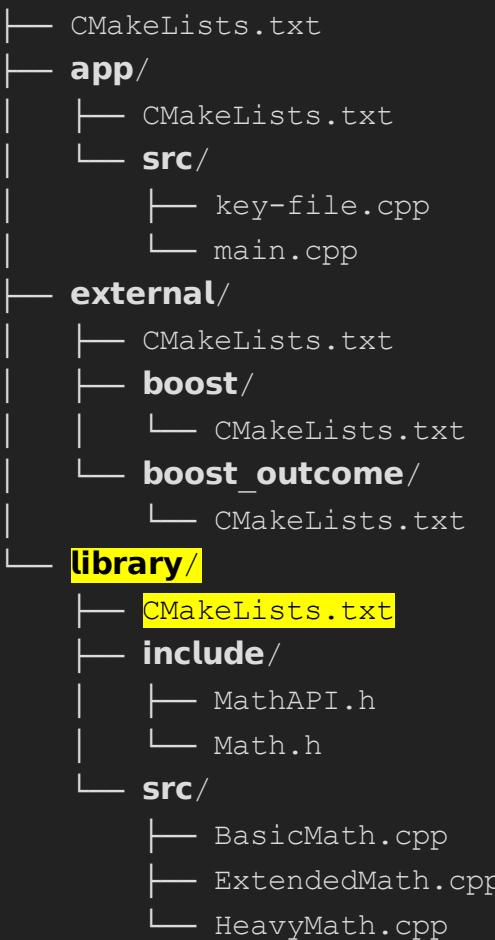
```
01 # ./library/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT )
12 target_sources( basicmath_ObjLib
13     PRIVATE    "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE   "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17 ...
```



ADDING BUILD/USAGE-REQUIREMENTS TO TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- `include_directories` adds include-search-paths on directory-scope
 - applies to all targets from that `CMakeLists.txt`

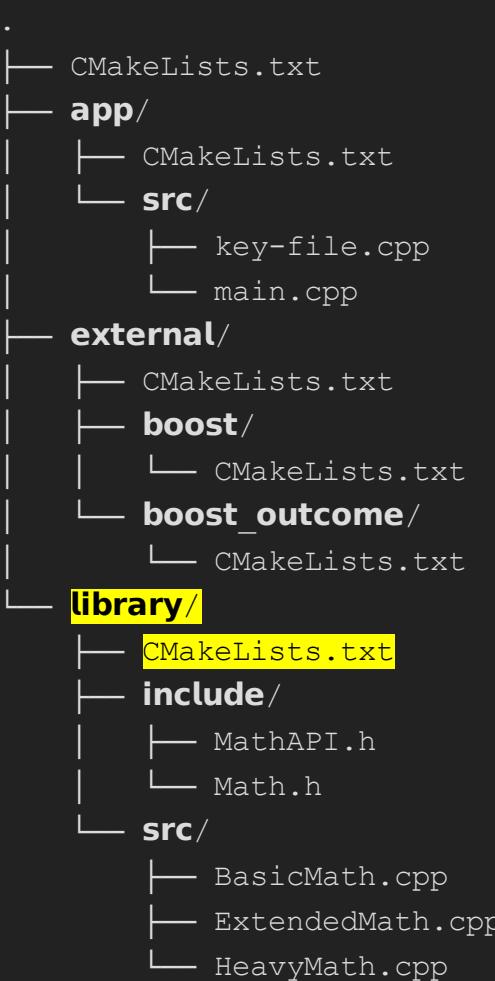


```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22 ...
```

ADDING BUILD/USAGE-REQUIREMENTS TO TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- `include_directories` adds include-search-paths on directory-scope
 - applies to all targets from that `CMakeLists.txt`
⇒ Not transitive, only sets *build-requirements*!

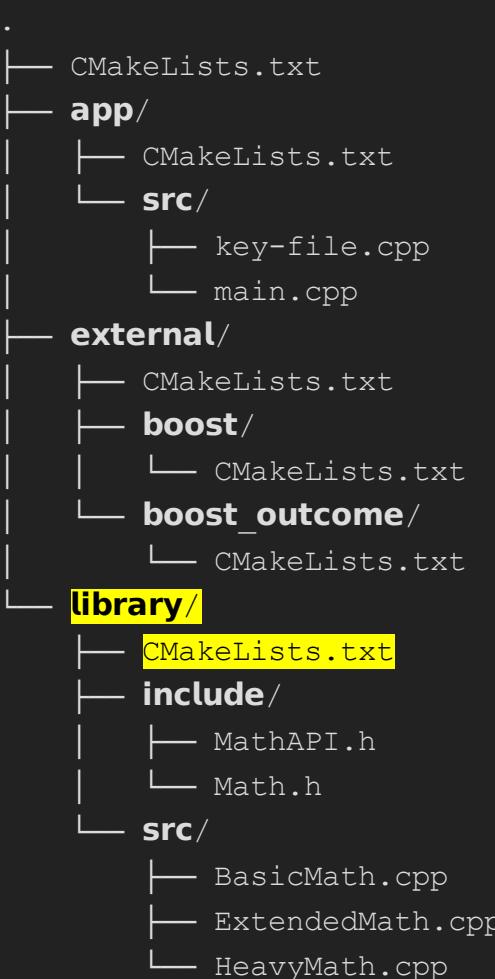


```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22 ...
```

ADDING BUILD/USAGE-REQUIREMENTS TO TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- `include_directories` adds include-search-paths on directory-scope
 - applies to all targets from that `CMakeLists.txt`
⇒ Not transitive, only sets *build-requirements*!
- Compile-features have to be set by hand.
 - set in `CMAKE_CXX_FLAGS` cache-variable

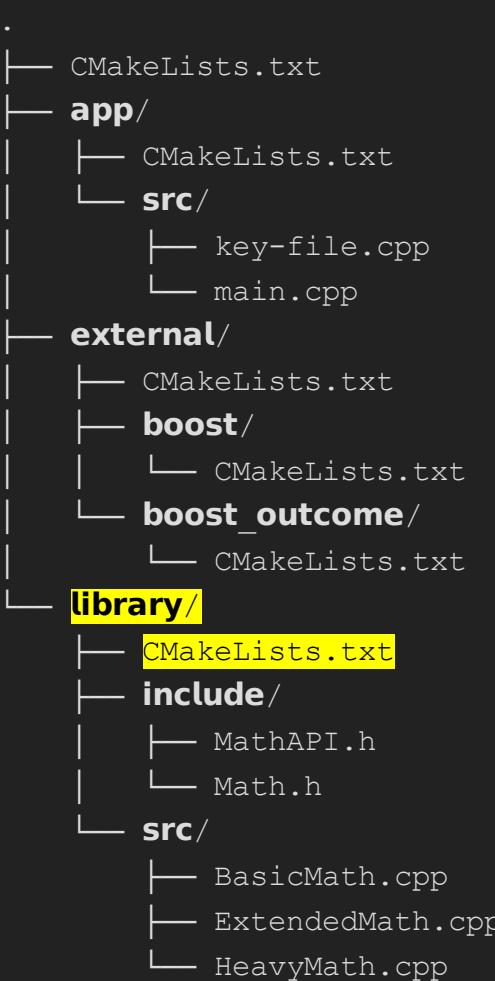


```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22 ...
```

ADDING BUILD/USAGE-REQUIREMENTS TO TARGETS

TRADITIONAL AND *Modern* CMAKE WAY

- `include_directories` adds include-search-paths on directory-scope
 - applies to all targets from that `CMakeLists.txt`
⇒ Not transitive, only sets *build-requirements*!
- Compile-features have to be set by hand.
 - set in `CMAKE_CXX_FLAGS` cache-variable
⇒ Not transitive, instead a **global build-requirement**!



```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22 ...
```

ADDING BUILD/USAGE-REQUIREMENTS TO TARGETS

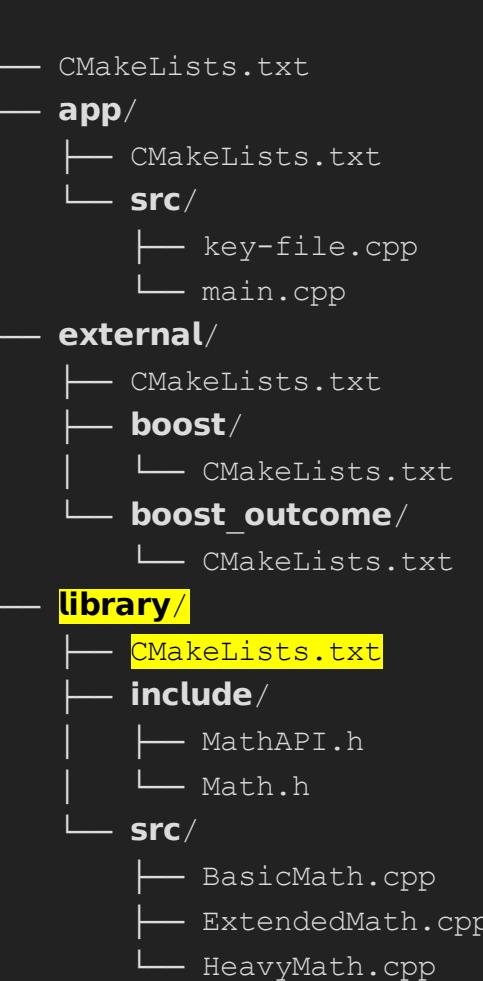
TRADITIONAL AND *Modern CMake* WAY

- `include_directories` adds include-search-paths on directory-scope
 - applies to all targets from that `CMakeLists.txt`
⇒ Not transitive, only sets *build-requirements*!
- Compile-features have to be set by hand.
 - set in `CMAKE_CXX_FLAGS` cache-variable
⇒ Not transitive, instead a **global build-requirement**!
- *Modern CMake* is more flexible (and explicit).
 - by using `target_include_directories` and `target_compile_features` commands.

```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22 ...
```



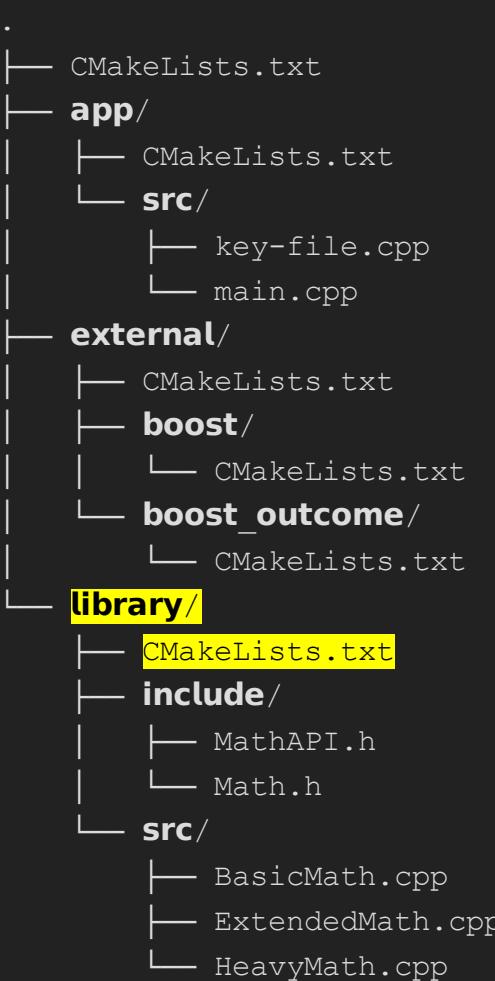
```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 ...
13 # Required include search-paths and constexpr support.
14 target_include_directories( basicmath_ObjLib
15     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
16 target_compile_features( basicmath_ObjLib
17     PUBLIC cxx_constexpr )
18 ...
```



ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

TRADITIONAL AND *Modern* CMAKE WAY

- *Usage-requirements* of dependencies need to become *build-requirements* of dependent targets using the same commands.

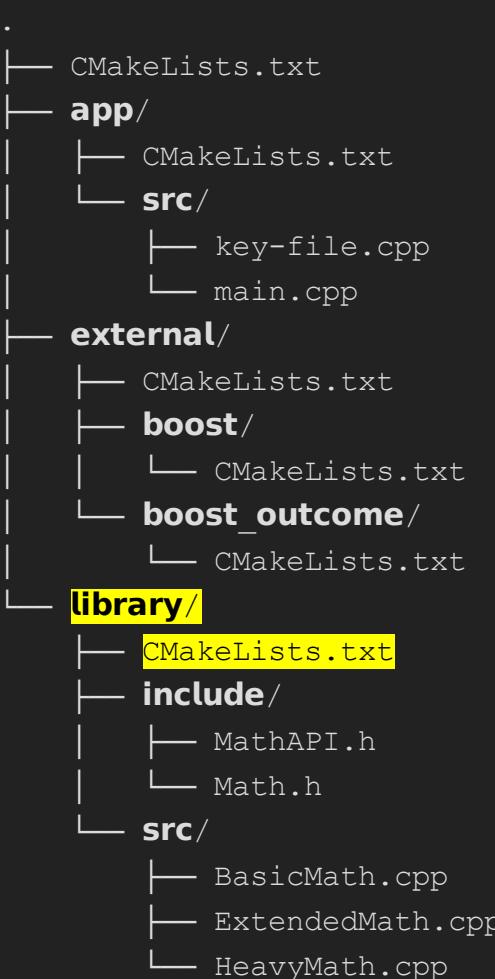


```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
29 ...
```

ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

TRADITIONAL AND *Modern* CMAKE WAY

- *Usage-requirements* of dependencies need to become *build-requirements* of dependent targets using the same commands.



```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
29 ...
```

ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

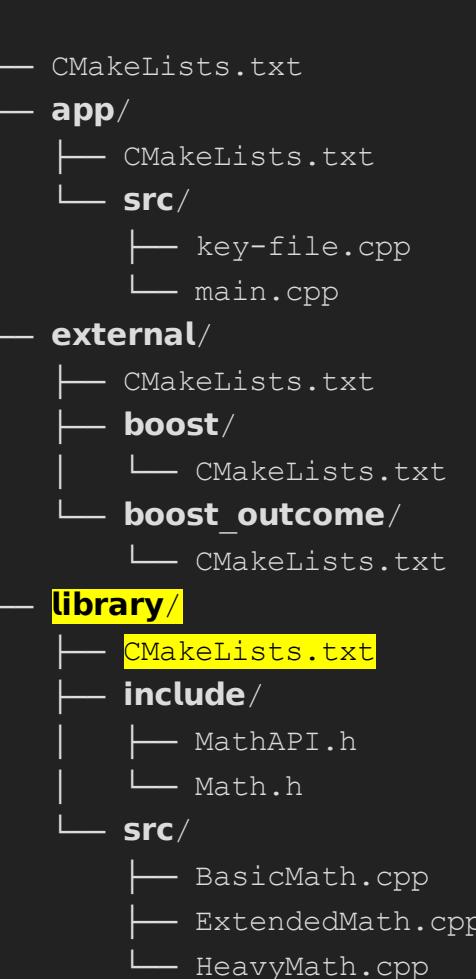
TRADITIONAL AND *Modern CMake* WAY

- *Usage-requirements* of dependencies need to become *build-requirements* of dependent targets using the same commands.
- Sadly, even with *Modern CMake*, **OBJECT** targets
 - cannot declare dependency on other targets, and therefore
 - need to add *usage-requirements* of dependencies by hand.

```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
29 ...
```



```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 ...
13 # Required include search-paths and constexpr support.
14 target_include_directories( basicmath_ObjLib
15     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
16 target_compile_features( basicmath_ObjLib
17     PUBLIC cxx_constexpr )
18
19 # Requires "Boost.Outcome" (which has some requirements, too).
20 target_include_directories( basicmath_ObjLib SYSTEM
21     PUBLIC
22         "$<TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES>" )
23 target_compile_features( basicmath_ObjLib
24     PUBLIC $<TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES> )
25 ...
```



ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

TRADITIONAL AND *Modern CMake* WAY

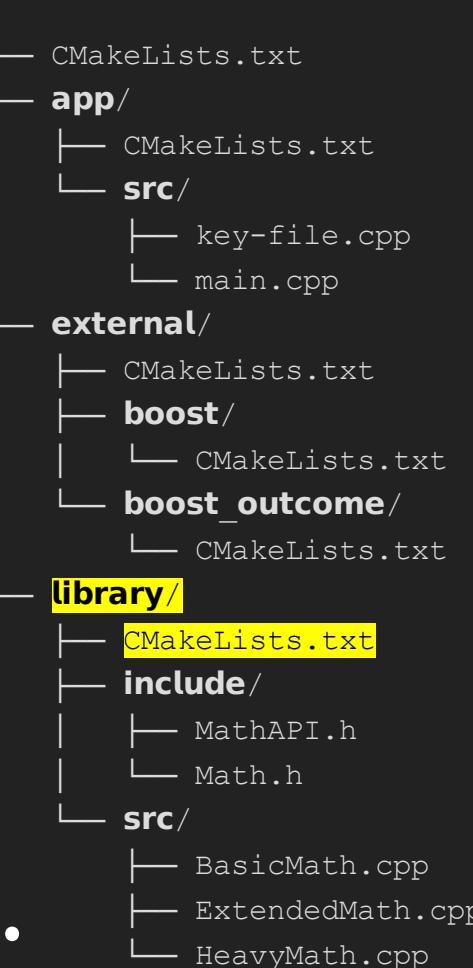
- *Usage-requirements* of dependencies need to become *build-requirements* of dependent targets using the same commands.
- Sadly, even with *Modern CMake*, **OBJECT** targets
 - cannot declare dependency on other targets, and therefore
 - need to add *usage-requirements* of dependencies by hand.

HOWEVER...

```
14 # ./library/CMakeLists.txt -- Traditional CMake
15 ...
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
29 ...
30 ...
```



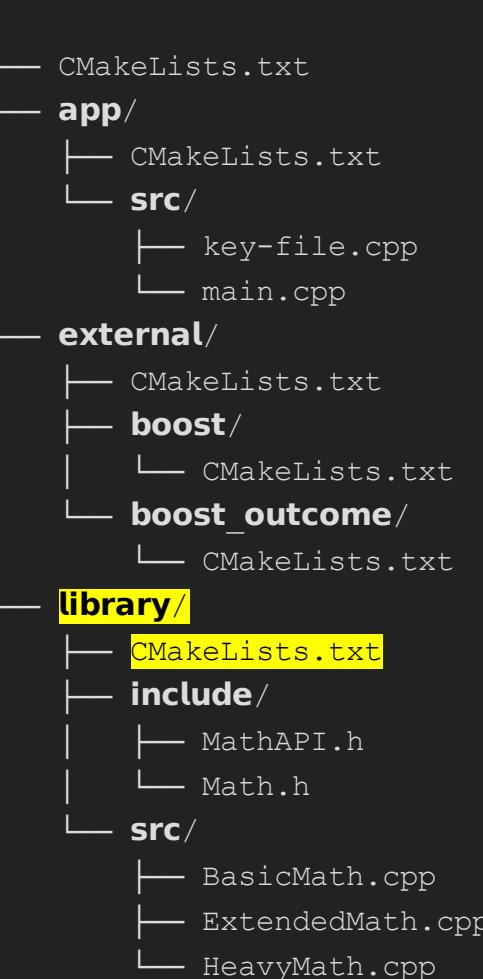
```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 ...
13 # Required include search-paths and constexpr support.
14 target_include_directories( basicmath_ObjLib
15     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
16 target_compile_features( basicmath_ObjLib
17     PUBLIC cxx_constexpr )
18
19 # Requires "Boost.Outcome" (which has some requirements, too).
20 target_include_directories( basicmath_ObjLib SYSTEM
21     PUBLIC
22         "$<TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES>" )
23 target_compile_features( basicmath_ObjLib
24     PUBLIC ${<TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES>} )
25 ...
26 ...
```



ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

MORE MODERN CMAKE WAY

- Since CMake 3.12 the `target_link_libraries` command can be used with OBJECT targets.



```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_include_directories( basicmath_ObjLib SYSTEM
26     PUBLIC
27         ${TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES} )
28 target_compile_features( basicmath_ObjLib
29     PUBLIC ${TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES} )
30 ...
```

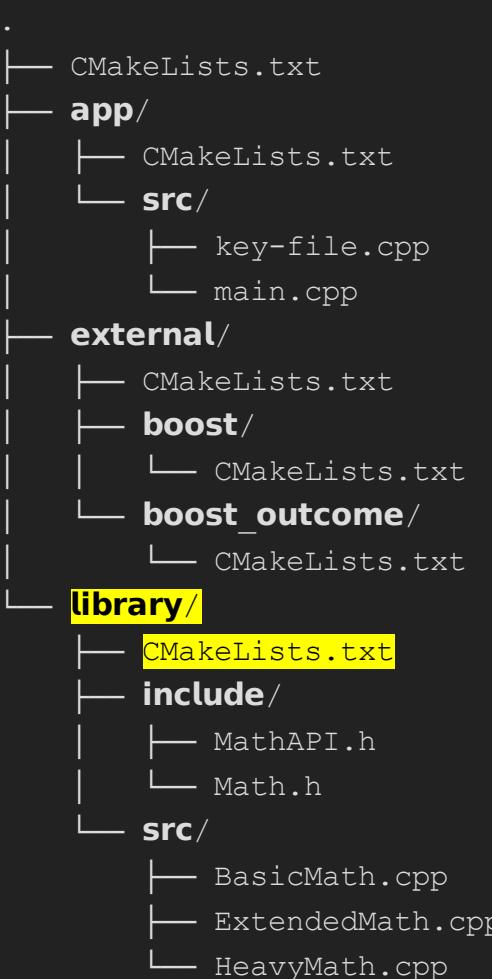


```
09 # ./library/CMakeLists.txt -- More Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_link_libraries( basicmath_ObjLib PUBLIC Boost::outcome )
26 ...
```

ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

MORE MODERN CMAKE WAY

- Since CMake 3.12 the `target_link_libraries` command can be used with OBJECT targets.
 - *Usage-requirements* of dependencies can be propagated to OBJECT targets.



```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_include_directories( basicmath_ObjLib SYSTEM
26     PUBLIC
27         ${TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES} )
28 target_compile_features( basicmath_ObjLib
29     PUBLIC ${TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES} )
30 ...
```



```
09 # ./library/CMakeLists.txt -- More Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_link_libraries( basicmath_ObjLib PUBLIC Boost::outcome )
26 ...
```

ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

MORE MODERN CMAKE WAY

- Since CMake 3.12 the `target_link_libraries` command can be used with OBJECT targets.
 - *Usage-requirements* of dependencies can be propagated to OBJECT targets.
 - But, the other way around, propagation of *usage-requirements* of OBJECT targets to other targets, is more interesting.

```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_include_directories( basicmath_ObjLib SYSTEM
26     PUBLIC
27         ${TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES} )
28 target_compile_features( basicmath_ObjLib
29     PUBLIC ${TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES} )
30 ...
```



```
09 # ./library/CMakeLists.txt -- More Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_link_libraries( basicmath_ObjLib PUBLIC Boost::outcome )
26 ...
```

```
.
  CMakeLists.txt
  app/
    CMakeLists.txt
    src/
      key-file.cpp
      main.cpp
  external/
    CMakeLists.txt
    boost/
      CMakeLists.txt
    boost_outcome/
      CMakeLists.txt
  library/
    CMakeLists.txt
    include/
      MathAPI.h
      Math.h
    src/
      BasicMath.cpp
      ExtendedMath.cpp
      HeavyMath.cpp
```

ADDING USAGE-REQUIREMENTS FROM DEPENDENCIES TO OBJECT TARGETS

MORE MODERN CMAKE WAY

- Since CMake 3.12 the `target_link_libraries` command can be used with OBJECT targets.
 - *Usage-requirements* of dependencies can be propagated to OBJECT targets.
 - But, the other way around, propagation of *usage-requirements* of OBJECT targets to other targets, is more interesting.

```
09 # ./library/CMakeLists.txt -- Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_include_directories( basicmath_ObjLib SYSTEM
26     PUBLIC
27         ${TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES} )
28 target_compile_features( basicmath_ObjLib
29     PUBLIC ${TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES} )
30 ...
```



```
09 # ./library/CMakeLists.txt -- More Modern CMake
10 ...
11 add_library( basicmath_ObjLib OBJECT )
17 ...
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_link_libraries( basicmath_ObjLib PUBLIC Boost::outcome )
26 ...
```

LET'S HAVE A LOOK...

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    └── boost/
        └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
library/
    ├── CMakeLists.txt
    └── include/
        ├── MathAPI.h
        └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

OBJECT LIBRARIES

SOME PECULIARITIES

LINKING OBJECT LIBRARY TARGETS – A QUIZ

```
01 # A common object-library target.
02 add_library( commonObjLib OBJECT )
03 target_sources( commonObjLib PRIVATE "common.cpp" )
04 target_include_directories( commonObjLib PUBLIC "include/common" )
05
06 # A shared-library target.
07 add_library( sharedLib1 SHARED )
08
09 # Another object-library target...
10 add_library( objLib2 OBJECT )
11 target_sources( objLib2 PRIVATE "source2.cpp" )
12 target_include_directories( objLib2 PUBLIC "include2" )
13 # ... and yet another one.
14 add_library( objLib3 OBJECT )
15 target_sources( objLib3 PRIVATE "source3.cpp" )
16 target_include_directories( objLib3 PUBLIC "include3" )
17
18 # Dependency on 'commonObjLib'.
19 target_link_libraries( sharedLib1 PUBLIC commonObjLib )
20 target_link_libraries( objLib2 PRIVATE commonObjLib )
21 target_link_libraries( objLib3 INTERFACE commonObjLib )
22
23 # Create executables linked against these targets.
24 add_executable( exe1 )
25 add_executable( exe2 )
26 add_executable( exe3 )
27 target_sources( exe1 PRIVATE "main1.cpp" )
28 target_sources( exe2 PRIVATE "main2.cpp" )
29 target_sources( exe3 PRIVATE "main3.cpp" )
30 target_link_libraries( exe1 PRIVATE sharedLib1 )
31 target_link_libraries( exe2 PRIVATE objLib2 )
32 target_link_libraries( exe3 PRIVATE objLib3 )
```

LINKING OBJECT LIBRARY TARGETS – A QUIZ

```
01 # A common object-library target.
02 add_library( commonObjLib OBJECT )
03 target_sources( commonObjLib PRIVATE "common.cpp" )
04 target_include_directories( commonObjLib PUBLIC "include/common" )
05
06 # A shared-library target.
07 add_library( sharedLib1 SHARED )
08
09 # Another object-library target...
10 add_library( objLib2 OBJECT )
11 target_sources( objLib2 PRIVATE "source2.cpp" )
12 target_include_directories( objLib2 PUBLIC "include2" )
13 # ... and yet another one.
14 add_library( objLib3 OBJECT )
15 target_sources( objLib3 PRIVATE "source3.cpp" )
16 target_include_directories( objLib3 PUBLIC "include3" )
17
18 # Dependency on 'commonObjLib'.
19 target_link_libraries( sharedLib1 PUBLIC commonObjLib )
20 target_link_libraries( objLib2 PRIVATE commonObjLib )
21 target_link_libraries( objLib3 INTERFACE commonObjLib )
22
23 # Create executables linked against these targets.
24 add_executable( exe1 )
25 add_executable( exe2 )
26 add_executable( exe3 )
27 target_sources( exe1 PRIVATE "main1.cpp" )
28 target_sources( exe2 PRIVATE "main2.cpp" )
29 target_sources( exe3 PRIVATE "main3.cpp" )
30 target_link_libraries( exe1 PRIVATE sharedLib1 )
31 target_link_libraries( exe2 PRIVATE objLib2 )
32 target_link_libraries( exe3 PRIVATE objLib3 )
```

- **Q:** What (local) include-directories does each individual target (`exe1`, `exe2` and `exe3`) know during compilation?

LINKING OBJECT LIBRARY TARGETS – A QUIZ

```
01 # A common object-library target.
02 add_library( commonObjLib OBJECT )
03 target_sources( commonObjLib PRIVATE "common.cpp" )
04 target_include_directories( commonObjLib PUBLIC "include/common" )
05
06 # A shared-library target.
07 add_library( sharedLib1 SHARED )
08
09 # Another object-library target...
10 add_library( objLib2 OBJECT )
11 target_sources( objLib2 PRIVATE "source2.cpp" )
12 target_include_directories( objLib2 PUBLIC "include2" )
13 # ... and yet another one.
14 add_library( objLib3 OBJECT )
15 target_sources( objLib3 PRIVATE "source3.cpp" )
16 target_include_directories( objLib3 PUBLIC "include3" )
17
18 # Dependency on 'commonObjLib'.
19 target_link_libraries( sharedLib1 PUBLIC commonObjLib )
20 target_link_libraries( objLib2 PRIVATE commonObjLib )
21 target_link_libraries( objLib3 INTERFACE commonObjLib )
22
23 # Create executables linked against these targets.
24 add_executable( exe1 )
25 add_executable( exe2 )
26 add_executable( exe3 )
27 target_sources( exe1 PRIVATE "main1.cpp" )
28 target_sources( exe2 PRIVATE "main2.cpp" )
29 target_sources( exe3 PRIVATE "main3.cpp" )
30 target_link_libraries( exe1 PRIVATE sharedLib1 )
31 target_link_libraries( exe2 PRIVATE objLib2 )
32 target_link_libraries( exe3 PRIVATE objLib3 )
```

- **Q:** What (local) include-directories does each individual target (`exe1`, `exe2` and `exe3`) know during compilation?

- **A:**

`exe1` — `.` and `include/common`

`exe2` — `.` and `include2`

`exe3` — `.` and `include3` and `include/common`

LINKING OBJECT LIBRARY TARGETS – A QUIZ

```
01 # A common object-library target.
02 add_library( commonObjLib OBJECT )
03 target_sources( commonObjLib PRIVATE "common.cpp" )
04 target_include_directories( commonObjLib PUBLIC "include/common" )
05
06 # A shared-library target.
07 add_library( sharedLib1 SHARED )
08
09 # Another object-library target...
10 add_library( objLib2 OBJECT )
11 target_sources( objLib2 PRIVATE "source2.cpp" )
12 target_include_directories( objLib2 PUBLIC "include2" )
13 # ... and yet another one.
14 add_library( objLib3 OBJECT )
15 target_sources( objLib3 PRIVATE "source3.cpp" )
16 target_include_directories( objLib3 PUBLIC "include3" )
17
18 # Dependency on 'commonObjLib'.
19 target_link_libraries( sharedLib1 PUBLIC commonObjLib )
20 target_link_libraries( objLib2 PRIVATE commonObjLib )
21 target_link_libraries( objLib3 INTERFACE commonObjLib )
22
23 # Create executables linked against these targets.
24 add_executable( exe1 )
25 add_executable( exe2 )
26 add_executable( exe3 )
27 target_sources( exe1 PRIVATE "main1.cpp" )
28 target_sources( exe2 PRIVATE "main2.cpp" )
29 target_sources( exe3 PRIVATE "main3.cpp" )
30 target_link_libraries( exe1 PRIVATE sharedLib1 )
31 target_link_libraries( exe2 PRIVATE objLib2 )
32 target_link_libraries( exe3 PRIVATE objLib3 )
```

- **Q:** What (local) include-directories does each individual target (`exe1`, `exe2` and `exe3`) know during compilation?

- **A:**

`exe1` – `.` and `include/common`

`exe2` – `.` and `include2`

`exe3` – `.` and `include3` and `include/common`

- **Q:** The generated output-binary of which executable target (`exe1`, `exe2` or `exe3`) contains the *object-files* for `common.cpp`?

LINKING OBJECT LIBRARY TARGETS – A QUIZ

```
01 # A common object-library target.
02 add_library( commonObjLib OBJECT )
03 target_sources( commonObjLib PRIVATE "common.cpp" )
04 target_include_directories( commonObjLib PUBLIC "include/common" )
05
06 # A shared-library target.
07 add_library( sharedLib1 SHARED )
08
09 # Another object-library target...
10 add_library( objLib2 OBJECT )
11 target_sources( objLib2 PRIVATE "source2.cpp" )
12 target_include_directories( objLib2 PUBLIC "include2" )
13 # ... and yet another one.
14 add_library( objLib3 OBJECT )
15 target_sources( objLib3 PRIVATE "source3.cpp" )
16 target_include_directories( objLib3 PUBLIC "include3" )
17
18 # Dependency on 'commonObjLib'.
19 target_link_libraries( sharedLib1 PUBLIC commonObjLib )
20 target_link_libraries( objLib2 PRIVATE commonObjLib )
21 target_link_libraries( objLib3 INTERFACE commonObjLib )
22
23 # Create executables linked against these targets.
24 add_executable( exe1 )
25 add_executable( exe2 )
26 add_executable( exe3 )
27 target_sources( exe1 PRIVATE "main1.cpp" )
28 target_sources( exe2 PRIVATE "main2.cpp" )
29 target_sources( exe3 PRIVATE "main3.cpp" )
30 target_link_libraries( exe1 PRIVATE sharedLib1 )
31 target_link_libraries( exe2 PRIVATE objLib2 )
32 target_link_libraries( exe3 PRIVATE objLib3 )
```

- **Q:** What (local) include-directories does each individual target (`exe1`, `exe2` and `exe3`) know during compilation?

- **A:**

`exe1` – `.` and `include/common`

`exe2` – `.` and `include2`

`exe3` – `.` and `include3` and `include/common`

- **Q:** The generated output-binary of which executable target (`exe1`, `exe2` or `exe3`) contains the *object-files* for `common.cpp`?

- **A:**

None! – Only the output-binary of `sharedLib` contains the object-files received from `commonObjLib`.

LINKING OBJECT LIBRARY TARGETS – A QUIZ

```
01 # A common object-library target.
02 add_library( commonObjLib OBJECT )
03 target_sources( commonObjLib PRIVATE "common.cpp" )
04 target_include_directories( commonObjLib PUBLIC "include/common" )
05
06 # A shared-library target.
07 add_library( sharedLib1 SHARED )
08
09 # Another object-library target...
10 add_library( objLib2 OBJECT )
11 target_sources( objLib2 PRIVATE "source2.cpp" )
12 target_include_directories( objLib2 PUBLIC "include2" )
13 # ... and yet another one.
14 add_library( objLib3 OBJECT )
15 target_sources( objLib3 PRIVATE "source3.cpp" )
16 target_include_directories( objLib3 PUBLIC "include3" )
17
18 # Dependency on 'commonObjLib'.
19 target_link_libraries( sharedLib1 PUBLIC commonObjLib )
20 target_link_libraries( objLib2 PRIVATE commonObjLib )
21 target_link_libraries( objLib3 INTERFACE commonObjLib )
22
23 # Create executables linked against these targets.
24 add_executable( exe1 )
25 add_executable( exe2 )
26 add_executable( exe3 )
27 target_sources( exe1 PRIVATE "main1.cpp" )
28 target_sources( exe2 PRIVATE "main2.cpp" )
29 target_sources( exe3 PRIVATE "main3.cpp" )
30 target_link_libraries( exe1 PRIVATE sharedLib1 )
31 target_link_libraries( exe2 PRIVATE objLib2 )
32 target_link_libraries( exe3 PRIVATE objLib3 )
```

- **Q:** What (local) include-directories does each individual target (`exe1`, `exe2` and `exe3`) know during compilation?

- **A:**

`exe1` – `.` and `include/common`

`exe2` – `.` and `include2`

`exe3` – `.` and `include3` and `include/common`

- **Q:** The generated output-binary of which executable target (`exe1`, `exe2` or `exe3`) contains the *object-files* for `common.cpp`?

- **A:**

None! – Only the output-binary of `sharedLib` contains the object-files received from `commonObjLib`.

Why is that so?

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF USAGE-REQUIREMENTS

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but not further propagating them to other targets.  
03 target_link_libraries( objTarget PRIVATE anyTarget )
```

```
01 # Not knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but further propagating them to other targets.  
03 target_link_libraries( objTarget INTERFACE anyTarget )
```

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # and further propagating them to other targets.  
03 target_link_libraries( objTarget PUBLIC anyTarget )
```

- *Usage-requirements* of any target (even `OBJECT` library targets) on the *right-hand-side* of `target_link_libraries` are propagated to the `OBJECT` library target on the *left-hand-side*.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF USAGE-REQUIREMENTS

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but not further propagating them to other targets.  
03 target_link_libraries( objTarget PRIVATE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *build-requirements* of `objTarget`

```
01 # Not knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but further propagating them to other targets.  
03 target_link_libraries( objTarget INTERFACE anyTarget )
```

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # and further propagating them to other targets.  
03 target_link_libraries( objTarget PUBLIC anyTarget )
```

- *Usage-requirements* of any target (even `OBJECT` library targets) on the *right-hand-side* of `target_link_libraries` are propagated to the `OBJECT` library target on the *left-hand-side*.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF USAGE-REQUIREMENTS

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but not further propagating them to other targets.  
03 target_link_libraries( objTarget PRIVATE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *build-requirements* of `objTarget`

```
01 # Not knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but further propagating them to other targets.  
03 target_link_libraries( objTarget INTERFACE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *usage-requirements* of `objTarget`

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # and further propagating them to other targets.  
03 target_link_libraries( objTarget PUBLIC anyTarget )
```

- *Usage-requirements* of any target (even `OBJECT` library targets) on the *right-hand-side* of `target_link_libraries` are propagated to the `OBJECT` library target on the *left-hand-side*.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF USAGE-REQUIREMENTS

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but not further propagating them to other targets.  
03 target_link_libraries( objTarget PRIVATE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *build-requirements* of `objTarget`

```
01 # Not knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but further propagating them to other targets.  
03 target_link_libraries( objTarget INTERFACE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *usage-requirements* of `objTarget`

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # and further propagating them to other targets.  
03 target_link_libraries( objTarget PUBLIC anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *usage/build-requirements* of `objTarget`

- *Usage-requirements* of any target (even `OBJECT` library targets) on the *right-hand-side* of `target_link_libraries` are propagated to the `OBJECT` library target on the *left-hand-side*.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF USAGE-REQUIREMENTS

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but not further propagating them to other targets.  
03 target_link_libraries( objTarget PRIVATE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *build-requirements* of `objTarget`

```
01 # Not knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # but further propagating them to other targets.  
03 target_link_libraries( objTarget INTERFACE anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *usage-requirements* of `objTarget`

```
01 # Knowing usage-requirements from 'anyTarget' when compiling to object-files,  
02 # and further propagating them to other targets.  
03 target_link_libraries( objTarget PUBLIC anyTarget )
```

- *usage-requirements* of `anyTarget` \Rightarrow *usage/build-requirements* of `objTarget`

- *Usage-requirements* of any target (even `OBJECT` library targets) on the *right-hand-side* of `target_link_libraries` are propagated to the `OBJECT` library target on the *left-hand-side*.
- **The same as for all other target types.**

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF OBJECT-FILES

```
01 # Object-files are only propagated to the build-requirements of another target
02 # if that other target is not an OBJECT library target itself!
03 target_link_libraries( nonObjTarget PRIVATE objTarget ) # Directly propagate object-files.
04 target_link_libraries( otherObjTarget PRIVATE objTarget ) # No propagation of object-files.
```

```
01 # No object-files can be propagated to the usage-requirements of any other target!
02 target_link_libraries( anyTarget INTERFACE objTarget ) # No indirect propagation of object-files.
```

```
01 # Object-files are propagated exactly as for PRIVATE propagation of object-files.
02 target_link_libraries( nonObjTarget PUBLIC objTarget ) # Directly propagate object-files.
03 target_link_libraries( otherObjTarget PUBLIC objTarget ) # No propagation of object-files.
```

- *Object-files* are only ever propagated to *direct* dependants!
 - And only, if that direct dependant is not an OBJECT library target itself.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF OBJECT-FILES

```
01 # Object-files are only propagated to the build-requirements of another target
02 # if that other target is not an OBJECT library target itself!
03 target_link_libraries( nonObjTarget PRIVATE objTarget ) # Directly propagate object-files.
04 target_link_libraries( otherObjTarget PRIVATE objTarget ) # No propagation of object-files.
```

- *object-files* of `objTarget` \Rightarrow *build-requirements* of `nonObjTarget`
- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `otherObjTarget`

```
01 # No object-files can be propagated to the usage-requirements of any other target!
02 target_link_libraries( anyTarget INTERFACE objTarget ) # No indirect propagation of object-files.
```

```
01 # Object-files are propagated exactly as for PRIVATE propagation of object-files.
02 target_link_libraries( nonObjTarget PUBLIC objTarget ) # Directly propagate object-files.
03 target_link_libraries( otherObjTarget PUBLIC objTarget ) # No propagation of object-files.
```

- *Object-files* are only ever propagated to *direct* dependants!
 - And only, if that direct dependant is not an OBJECT library target itself.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF OBJECT-FILES

```
01 # Object-files are only propagated to the build-requirements of another target
02 # if that other target is not an OBJECT library target itself!
03 target_link_libraries( nonObjTarget PRIVATE objTarget ) # Directly propagate object-files.
04 target_link_libraries( otherObjTarget PRIVATE objTarget ) # No propagation of object-files.
```

- *object-files* of `objTarget` \Rightarrow *build-requirements* of `nonObjTarget`
- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `otherObjTarget`

```
01 # No object-files can be propagated to the usage-requirements of any other target!
02 target_link_libraries( anyTarget INTERFACE objTarget ) # No indirect propagation of object-files.
```

- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `anyTarget`

```
01 # Object-files are propagated exactly as for PRIVATE propagation of object-files.
02 target_link_libraries( nonObjTarget PUBLIC objTarget ) # Directly propagate object-files.
03 target_link_libraries( otherObjTarget PUBLIC objTarget ) # No propagation of object-files.
```

- *Object-files* are only ever propagated to *direct* dependants!
 - And only, if that direct dependant is not an `OBJECT` library target itself.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF OBJECT-FILES

```
01 # Object-files are only propagated to the build-requirements of another target
02 # if that other target is not an OBJECT library target itself!
03 target_link_libraries( nonObjTarget PRIVATE objTarget ) # Directly propagate object-files.
04 target_link_libraries( otherObjTarget PRIVATE objTarget ) # No propagation of object-files.
```

- *object-files* of `objTarget` \Rightarrow *build-requirements* of `nonObjTarget`
- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `otherObjTarget`

```
01 # No object-files can be propagated to the usage-requirements of any other target!
02 target_link_libraries( anyTarget INTERFACE objTarget ) # No indirect propagation of object-files.
```

- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `anyTarget`

```
01 # Object-files are propagated exactly as for PRIVATE propagation of object-files.
02 target_link_libraries( nonObjTarget PUBLIC objTarget ) # Directly propagate object-files.
03 target_link_libraries( otherObjTarget PUBLIC objTarget ) # No propagation of object-files.
```

- *object-files* of `objTarget` \Rightarrow *build-requirements* of `nonObjTarget`
- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `otherObjTarget`
- *Object-files* are only ever propagated to *direct* dependants!
 - And only, if that direct dependant is not an OBJECT library target itself.

RULES FOR LINKING OBJECT LIBRARY TARGETS

PROPAGATION OF OBJECT-FILES

```
01 # Object-files are only propagated to the build-requirements of another target
02 # if that other target is not an OBJECT library target itself!
03 target_link_libraries( nonObjTarget PRIVATE objTarget ) # Directly propagate object-files.
04 target_link_libraries( otherObjTarget PRIVATE objTarget ) # No propagation of object-files.
```

- *object-files* of `objTarget` \Rightarrow *build-requirements* of `nonObjTarget`
- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `otherObjTarget`

```
01 # No object-files can be propagated to the usage-requirements of any other target!
02 target_link_libraries( anyTarget INTERFACE objTarget ) # No indirect propagation of object-files.
```

- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `anyTarget`

```
01 # Object-files are propagated exactly as for PRIVATE propagation of object-files.
02 target_link_libraries( nonObjTarget PUBLIC objTarget ) # Directly propagate object-files.
03 target_link_libraries( otherObjTarget PUBLIC objTarget ) # No propagation of object-files.
```

- *object-files* of `objTarget` \Rightarrow *build-requirements* of `nonObjTarget`
- *object-files* of `objTarget` $\not\Rightarrow$ *usage/build-requirements* of `otherObjTarget`
 - *Object-files are only ever propagated to direct dependants!*
 - And only, if that direct dependant is not an OBJECT library target itself.

CREATING LIBRARIES (CONT.)

BASICMATH / EXTMATH

CREATING LIBRARIES – AN OVERVIEW (CONT.)

TRADITIONAL CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
```

```
29
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36
37 # A shared library for advanced-math functionality.
38 add_library( extmath SHARED
39     "src/ExtendedMath.cpp" # Premium-content!
40     $<TARGET_OBJECTS:basicmath_ObjLib> )
41 # The usage-requirements of the OBJECT-library (and its
42 # direct dependency) are already set in directory scope
43 # (above in lines 19 to 28).
44
45 # Target 'extmath' requires include-path and needs
46 # to link to library-file of its extra dependency
47 # "Boost.Graph".
48 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
49 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
library/
    └── CMakeLists.txt
include/
    ├── MathAPI.h
    └── Math.h
src/
    ├── BasicMath.cpp
    ├── ExtendedMath.cpp
    └── HeavyMath.cpp
```

CREATING LIBRARIES – AN OVERVIEW (CONT.)

TRADITIONAL CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
```

```
29
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36
37 # A shared library for advanced-math functionality.
38 add_library( extmath SHARED
39     "src/ExtendedMath.cpp" # Premium-content!
40     $<TARGET_OBJECTS:basicmath_ObjLib> )
41 # The usage-requirements of the OBJECT-library (and its
42 # direct dependency) are already set in directory scope
43 # (above in lines 19 to 28).
44
45 # Target 'extmath' requires include-path and needs
46 # to link to library-file of its extra dependency
47 # "Boost.Graph".
48 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
49 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

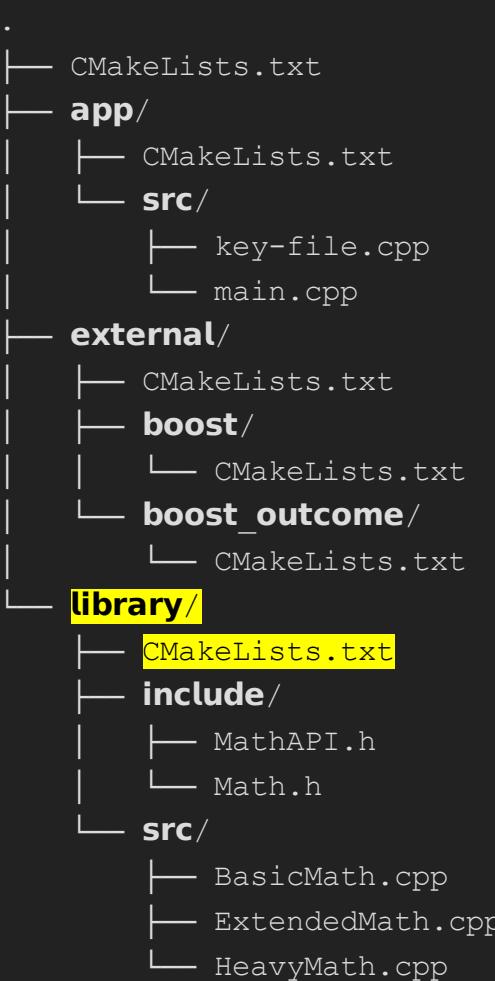
```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    └── boost/
        └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
    └── library/
        └── CMakeLists.txt
    └── include/
        ├── MathAPI.h
        └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

LET'S CONTINUE STEP BY STEP...

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- Object-files from the `OBJECT` target are the *sources* of the `SHARED` library target.



```
28 # ./library/CMakeLists.txt -- Traditional CMake
29 ...
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
```

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- Object-files from the `OBJECT` target are the *sources* of the `SHARED` library target.

```
.
  └── CMakeLists.txt
  └── app/
    └── CMakeLists.txt
      └── src/
        └── key-file.cpp
        └── main.cpp
  └── external/
    └── CMakeLists.txt
  └── boost/
    └── CMakeLists.txt
  └── boost_outcome/
    └── CMakeLists.txt
  └── library/
    └── CMakeLists.txt
  └── include/
    └── MathAPI.h
    └── Math.h
  └── src/
    └── BasicMath.cpp
    └── ExtendedMath.cpp
    └── HeavyMath.cpp
```

```
28 # ./library/CMakeLists.txt -- Traditional CMake
29 ...
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
```

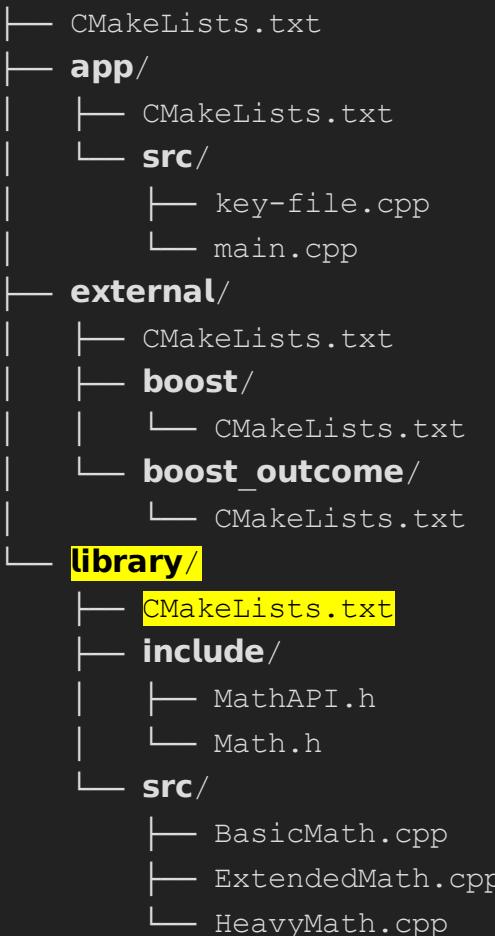


```
29 # ./library/CMakeLists.txt -- Modern CMake
30 ...
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE $<TARGET_OBJECTS:basicmath_ObjLib> )
```

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- Object-files from the `OBJECT` target are the *sources* of the `SHARED` library target.
- *Traditional CMake*:
The *usage-requirements* of the `OBJECT` library were set on directory-scope
 - and are therefore already set on the `SHARED` library target.



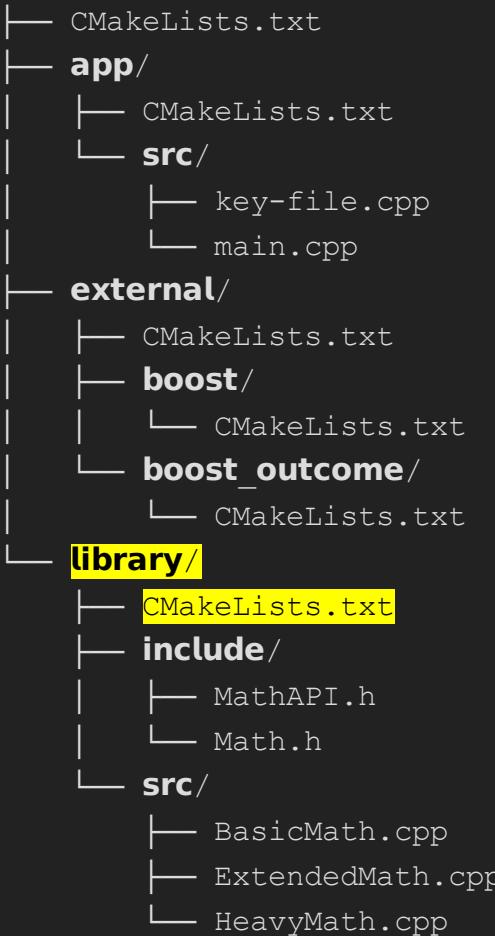
```
28 # ./library/CMakeLists.txt -- Traditional CMake
29 ...
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36 ...
```

```
29 # ./library/CMakeLists.txt -- Modern CMake
30 ...
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE $<TARGET_OBJECTS:basicmath_ObjLib> )
```

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND *Modern CMake* WAY

- Object-files from the `OBJECT` target are the *sources* of the `SHARED` library target.
- *Traditional CMake*:
The *usage-requirements* of the `OBJECT` library were set on directory-scope
 - and are therefore already set on the `SHARED` library target.
- *Modern CMake* needs to propagate *usage-requirements* explicitly.



```
28 # ./library/CMakeLists.txt -- Traditional CMake
29 ...
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36 ...
```

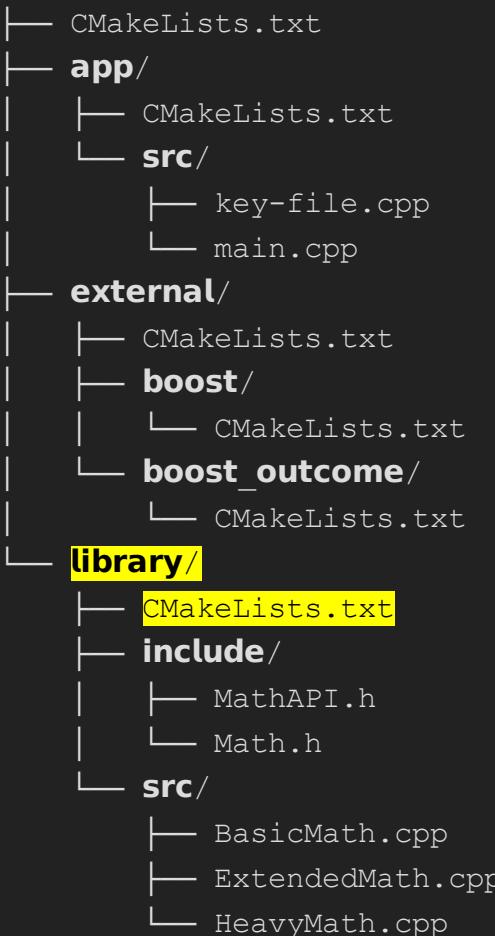


```
29 # ./library/CMakeLists.txt -- Modern CMake
30 ...
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE $<TARGET_OBJECTS:basicmath_ObjLib> )
35 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
36 # (and its indirect dependency "Boost.Outcome").
37 target_include_directories( basicmath
38     PUBLIC
39         $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES> )
40 target_compile_features( basicmath
41     PUBLIC $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES> )
42 ...
```

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND *Modern CMake* WAY

- Object-files from the `OBJECT` target are the *sources* of the `SHARED` library target.
- *Traditional CMake*:
The *usage-requirements* of the `OBJECT` library were set on directory-scope
 - and are therefore already set on the `SHARED` library target.
- *Modern CMake* needs to propagate *usage-requirements* explicitly.



```
28 # ./library/CMakeLists.txt -- Traditional CMake
29 ...
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36 ...
```



```
29 # ./library/CMakeLists.txt -- Modern CMake
30 ...
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE $<TARGET_OBJECTS:basicmath_ObjLib> )
35 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
36 # (and its indirect dependency "Boost.Outcome").
37 target_include_directories( basicmath
38     PUBLIC
39         $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES> )
40 target_compile_features( basicmath
41     PUBLIC $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES> )
42 ...
```

HOWEVER (AS YOU ALREADY KNOW)...

CREATING TARGETS FROM OBJECT TARGETS

MORE MODERN CMAKE WAY

- Since CMake 3.12 the `target_link_libraries` command can be used with OBJECT targets (as you have seen already).
 - *Object-files* files are propagated to direct dependants (only).
 - *Usage-requirements* are propagated as for all other targets.

```
29 # ./library/CMakeLists.txt -- Modern CMake
30 ...
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE $<TARGET_OBJECTS:basicmath_ObjLib> )
35 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
36 # (and its indirect dependency "Boost.Outcome").
37 target_include_directories( basicmath
38     PUBLIC
39         $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES> )
40 target_compile_features( basicmath
41     PUBLIC $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES> )
42 ...
```



```
25 # ./library/CMakeLists.txt -- More Modern CMake
26 ...
27 # A shared library for basic-math functionality.
28 add_library( basicmath SHARED )
29 target_link_libraries( basicmath PUBLIC basicmath_ObjLib )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    └── boost/
        └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
└── library/
    └── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND MODERN CMAKE WAY

- The same applies to the second library.
 - Additionally, a dependency to *Boost.Graph* is needed.

```
34 # ./library/CMakeLists.txt -- Traditional CMake
35 ...
36 # A shared library for advanced-math functionality.
37 add_library( extmath SHARED
38   "src/ExtendedMath.cpp"  # Premium-content!
39   ${TARGET_OBJECTS:basicmath_ObjLib} )
40 # The usage-requirements of the OBJECT-library (and its
41 # direct dependency) are already set in directory scope
42 # (above in lines 19 to 28).
43
44 # Target 'extmath' requires include-path and needs
45 # to link to library-file of its extra dependency
46 # "Boost.Graph".
47 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
48 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

```
.
  CMakeLists.txt
  app/
    CMakeLists.txt
    src/
      key-file.cpp
      main.cpp
  external/
    CMakeLists.txt
    boost/
      CMakeLists.txt
    boost_outcome/
      CMakeLists.txt
  library/
    CMakeLists.txt
    include/
      MathAPI.h
      Math.h
    src/
      BasicMath.cpp
      ExtendedMath.cpp
      HeavyMath.cpp
```

CREATING TARGETS FROM OBJECT TARGETS

TRADITIONAL AND *Modern CMake* WAY

- The same applies to the second library.
 - Additionally, a dependency to *Boost.Graph* is needed.

```
34 # ./library/CMakeLists.txt -- Traditional CMake
35 ...
36 # A shared library for advanced-math functionality.
37 add_library( extmath SHARED
38     "src/ExtendedMath.cpp" # Premium-content!
39     ${TARGET_OBJECTS:basicmath_ObjLib} )
40 # The usage-requirements of the OBJECT-library (and its
41 # direct dependency) are already set in directory scope
42 # (above in lines 19 to 28).
43
44 # Target 'extmath' requires include-path and needs
45 # to link to library-file of its extra dependency
46 # "Boost.Graph".
47 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
48 target_link_libraries( extmath ${Boost_LIBRARIES} )
```



```
41 # ./library/CMakeLists.txt -- Modern CMake
42 ...
43 # A shared library for advanced-math functionality.
44 add_library( extmath SHARED "src/dummy.cpp" )
45 target_sources( extmath
46     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
47     ${TARGET_OBJECTS:basicmath_ObjLib} )
48 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
49 # (and its indirect dependency "Boost.Outcome").
50 target_include_directories( extmath
51     PUBLIC
52     ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES} )
53 target_compile_features( extmath
54     PUBLIC ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES} )
55 # "Boost.Graph" is an additional dependency.
56 target_link_libraries( extmath PRIVATE Boost::graph )
57
58 # Create ALIAS targets.
59 add_library( MyCalc::basicmath ALIAS basicmath )
60 add_library( MyCalc::extmath ALIAS extmath )
```

```
.
├── CMakeLists.txt
├── app/
│   └── CMakeLists.txt
│       └── src/
│           └── key-file.cpp
│           └── main.cpp
└── external/
    └── CMakeLists.txt
        └── boost/
            └── CMakeLists.txt
        └── boost_outcome/
            └── CMakeLists.txt
    └── library/
        └── CMakeLists.txt
        └── include/
            └── MathAPI.h
            └── Math.h
        └── src/
            └── BasicMath.cpp
            └── ExtendedMath.cpp
            └── HeavyMath.cpp
```

CREATING TARGETS FROM OBJECT TARGETS

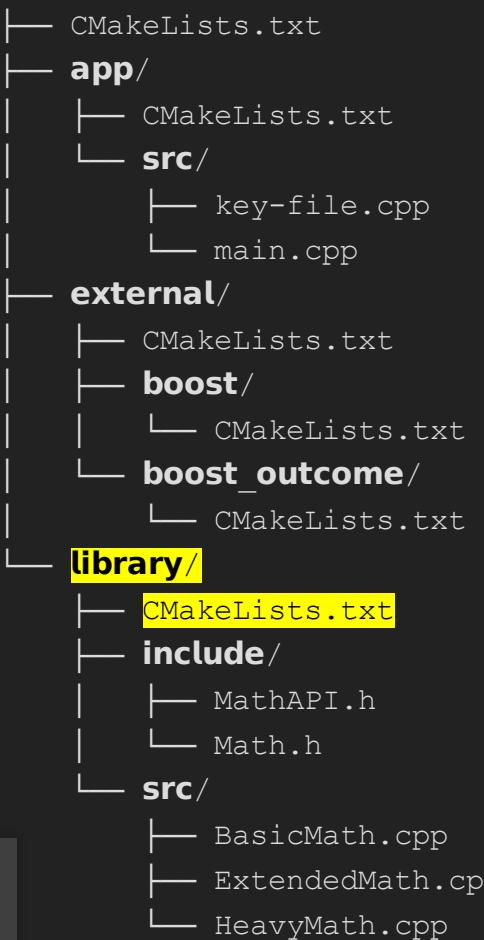
TRADITIONAL AND *Modern CMake* WAY

- The same applies to the second library.
 - Additionally, a dependency to *Boost.Graph* is needed.
 - Aliases might come in handy, too.

```
34 # ./library/CMakeLists.txt -- Traditional CMake
35 ...
36 # A shared library for advanced-math functionality.
37 add_library( extmath SHARED
38     "src/ExtendedMath.cpp" # Premium-content!
39     ${TARGET_OBJECTS:basicmath_ObjLib} )
40 # The usage-requirements of the OBJECT-library (and its
41 # direct dependency) are already set in directory scope
42 # (above in lines 19 to 28).
43
44 # Target 'extmath' requires include-path and needs
45 # to link to library-file of its extra dependency
46 # "Boost.Graph".
47 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
48 target_link_libraries( extmath ${Boost_LIBRARIES} )
```



```
41 # ./library/CMakeLists.txt -- Modern CMake
42 ...
43 # A shared library for advanced-math functionality.
44 add_library( extmath SHARED "src/dummy.cpp" )
45 target_sources( extmath
46     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
47     ${TARGET_OBJECTS:basicmath_ObjLib} )
48 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
49 # (and its indirect dependency "Boost.Outcome").
50 target_include_directories( extmath
51     PUBLIC
52     ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES} )
53 target_compile_features( extmath
54     PUBLIC ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES} )
55 # "Boost.Graph" is an additional dependency.
56 target_link_libraries( extmath PRIVATE Boost::graph )
57
58 # Create ALIAS targets.
59 add_library( MyCalc::basicmath ALIAS basicmath )
60 add_library( MyCalc::extmath ALIAS extmath )
```



CREATING TARGETS FROM OBJECT TARGETS

MORE MODERN CMAKE WAY

```
41 # ./library/CMakeLists.txt -- Modern CMake
42 ...
43 # A shared library for advanced-math functionality.
44 add_library( extmath SHARED "src/dummy.cpp" )
45 target_sources( extmath
46     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
47     $<TARGET_OBJECTS:basicmath_ObjLib> )
48 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
49 # (and its indirect dependency "Boost.Outcome").
50 target_include_directories( extmath
51     PUBLIC
52     $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES> )
53 target_compile_features( extmath
54     PUBLIC $<TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES> )
55 # "Boost.Graph" is an additional dependency.
56 target_link_libraries( extmath PRIVATE Boost::graph )
57
58 # Create ALIAS targets.
59 add_library( MyCalc::basicmath ALIAS basicmath )
60 add_library( MyCalc::extmath ALIAS extmath )
```



- **With CMake 3.12:**

```
29 # ./library/CMakeLists.txt -- More Modern CMake
30 ...
31 # A shared library for advanced-math functionality.
32 add_library( extmath SHARED )
33 target_sources( extmath
34     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
35 target_link_libraries( extmath PUBLIC basicmath_ObjLib )
36 # "Boost.Graph" is an additional dependency.
37 target_link_libraries( extmath PRIVATE Boost::graph )
38
39 # Create ALIAS targets.
40 add_library( MyCalc::basicmath ALIAS basicmath )
41 add_library( MyCalc::extmath ALIAS extmath )
```

```
.   CMakeLists.txt
  app/   CMakeLists.txt
    src/   key-file.cpp
          main.cpp
  external/ CMakeLists.txt
    boost/   CMakeLists.txt
      boost_outcome/ CMakeLists.txt
  library/ CMakeLists.txt
    include/ MathAPI.h
      Math.h
    src/   BasicMath.cpp
          ExtendedMath.cpp
          HeavyMath.cpp
```

CREATING LIBRARIES – AN OVERVIEW (CONT.)

TRADITIONAL CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
```

```
29
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36
37 # A shared library for advanced-math functionality.
38 add_library( extmath SHARED
39     "src/ExtendedMath.cpp" # Premium-content!
40     $<TARGET_OBJECTS:basicmath_ObjLib> )
41 # The usage-requirements of the OBJECT-library (and its
42 # direct dependency) are already set in directory scope
43 # (above in lines 19 to 28).
44
45 # Target 'extmath' requires include-path and needs
46 # to link to library-file of its extra dependency
47 # "Boost.Graph".
48 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
49 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
library/
    └── CMakeLists.txt
include/
    ├── MathAPI.h
    └── Math.h
src/
    ├── BasicMath.cpp
    ├── ExtendedMath.cpp
    └── HeavyMath.cpp
```

CREATING LIBRARIES – AN OVERVIEW (CONT.)

TRADITIONAL CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( MathLibs )
06 set( VERSION 1.0.0 )
07 set( DESCRIPTION "The internal math-libraries." )
08
09 # Sources for common functionality, used in both math-libraries.
10 set( BASIC_SOURCES
11     "src/BasicMath.cpp"
12     "src/HeavyMath.cpp" # Takes loooooong to compile!
13     "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
14     "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
15 # An OBJECT-library, used to only compile common sources once!
16 add_library( basicmath_ObjLib OBJECT ${BASIC_SOURCES} )
17
18 # Required include search-paths and constexpr support.
19 include_directories( "${CMAKE_CURRENT_SOURCE_DIR}/include" )
20 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11" CACHE
21     STRING "C++ compile-flags" FORCE )
22
23 # Requires "Boost.Outcome" (which has some requirements, too).
24 include_directories( SYSTEM
25     "$<TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS>" )
26 get_target_property( props Boost::outcome MY_COMPILE_FEATURES )
27 set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${props}" CACHE
28     STRING "C++ compile-flags" FORCE )
```

```
29
30 # A shared library for basic-math functionality.
31 add_library( basicmath SHARED
32     $<TARGET_OBJECTS:basicmath_ObjLib> )
33 # The usage-requirements of the OBJECT-library (and its
34 # direct dependency) are already set in directory scope
35 # (above in lines 19 to 28).
36
37 # A shared library for advanced-math functionality.
38 add_library( extmath SHARED
39     "src/ExtendedMath.cpp" # Premium-content!
40     $<TARGET_OBJECTS:basicmath_ObjLib> )
41 # The usage-requirements of the OBJECT-library (and its
42 # direct dependency) are already set in directory scope
43 # (above in lines 19 to 28).
44
45 # Target 'extmath' requires include-path and needs
46 # to link to library-file of its extra dependency
47 # "Boost.Graph".
48 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
49 target_link_libraries( extmath ${Boost_LIBRARIES} )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
library/
    └── CMakeLists.txt
include/
    ├── MathAPI.h
    └── Math.h
src/
    ├── BasicMath.cpp
    ├── ExtendedMath.cpp
    └── HeavyMath.cpp
```

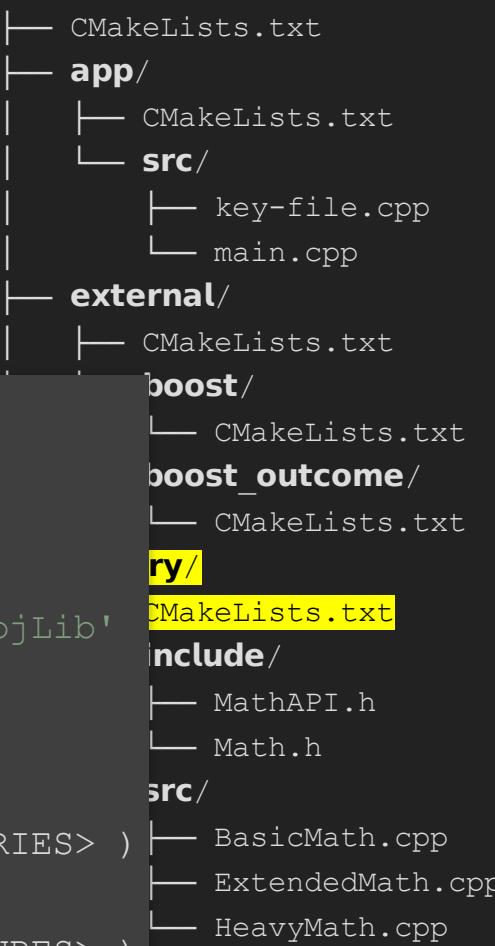


CREATING LIBRARIES – AN OVERVIEW (CONT.)

MODERN CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13     PRIVATE "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_include_directories( basicmath_ObjLib SYSTEM
26     PUBLIC
27     ${TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES} )
28 target_compile_features( basicmath_ObjLib
29     PUBLIC ${TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES} )
30
```

```
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE ${TARGET_OBJECTS:basicmath_ObjLib} )
35 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
36 # (and its indirect dependency "Boost.Outcome").
37 target_include_directories( basicmath
38     PUBLIC
39     ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES} )
40 target_compile_features( basicmath
41     PUBLIC ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES} )
42
43 # A shared library for advanced-math functionality.
44 add_library( extmath SHARED "src/dummy.cpp" )
45 target_sources( extmath
46     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
47     ${TARGET_OBJECTS:basicmath_ObjLib} )
48 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
49 # (and its indirect dependency "Boost.Outcome").
50 target_include_directories( extmath
51     PUBLIC
52     ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES} )
53 target_compile_features( extmath
54     PUBLIC ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES} )
55 # "Boost.Graph" is an additional dependency.
56 target_link_libraries( extmath PRIVATE Boost::graph )
57
58 # Create ALIAS targets.
59 add_library( MyCalc::basicmath ALIAS basicmath )
60 add_library( MyCalc::extmath ALIAS extmath )
```

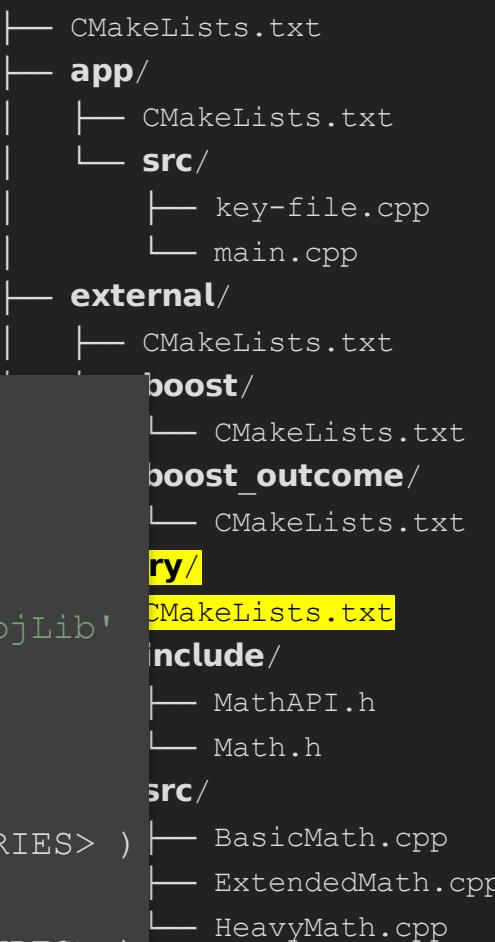


CREATING LIBRARIES – AN OVERVIEW (CONT.)

MODERN CMAKE WAY

```
01 # ./library/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT "src/dummy.cpp" )
12 target_sources( basicmath_ObjLib
13     PRIVATE "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_include_directories( basicmath_ObjLib SYSTEM
26     PUBLIC
27     ${TARGET_PROPERTY:Boost::outcome,INTERFACE_INCLUDE_DIRECTORIES} )
28 target_compile_features( basicmath_ObjLib
29     PUBLIC ${TARGET_PROPERTY:Boost::outcome,INTERFACE_COMPILE_FEATURES} )
30
```

```
31 # A shared library for basic-math functionality.
32 add_library( basicmath SHARED "src/dummy.cpp" )
33 target_sources( basicmath
34     PRIVATE ${TARGET_OBJECTS:basicmath_ObjLib} )
35 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
36 # (and its indirect dependency "Boost.Outcome").
37 target_include_directories( basicmath
38     PUBLIC
39     ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES} )
40 target_compile_features( basicmath
41     PUBLIC ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES} )
42
43 # A shared library for advanced-math functionality.
44 add_library( extmath SHARED "src/dummy.cpp" )
45 target_sources( extmath
46     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
47     ${TARGET_OBJECTS:basicmath_ObjLib} )
48 # Inherit the usage-requirements from direct dependency 'basicmath_ObjLib'
49 # (and its indirect dependency "Boost.Outcome").
50 target_include_directories( extmath
51     PUBLIC
52     ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_INCLUDE_DIRECTORIES} )
53 target_compile_features( extmath
54     PUBLIC ${TARGET_PROPERTY:basicmath_ObjLib,INTERFACE_COMPILE_FEATURES} )
55 # "Boost.Graph" is an additional dependency.
56 target_link_libraries( extmath PRIVATE Boost::graph )
57
58 # Create ALIAS targets.
59 add_library( MyCalc::basicmath ALIAS basicmath )
60 add_library( MyCalc::extmath ALIAS extmath )
```



CREATING LIBRARIES – AN OVERVIEW (CONT.)

MORE MODERN CMAKE WAY

```
01 # ./library/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.12 )
04
05 project( MathLibs
06     VERSION 1.0.0
07     DESCRIPTION "The internal math-libraries." )
08
09 # An OBJECT-library, used to only compile common sources once
10 # which are used in both math-libraries.
11 add_library( basicmath_ObjLib OBJECT )
12 target_sources( basicmath_ObjLib
13     PRIVATE "src/BasicMath.cpp"
14     "src/HeavyMath.cpp" # Takes loooooong to compile!
15     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include/Math.h"
16     INTERFACE "${CMAKE_CURRENT_SOURCE_DIR}/include/MathAPI.h" )
17
18 # Required include search-paths and constexpr support.
19 target_include_directories( basicmath_ObjLib
20     PUBLIC "${CMAKE_CURRENT_SOURCE_DIR}/include" )
21 target_compile_features( basicmath_ObjLib
22     PUBLIC cxx_constexpr )
23
24 # Requires "Boost.Outcome" (which has some requirements, too).
25 target_link_libraries( basicmath_ObjLib PUBLIC Boost::outcome )
```

```
26
27 # A shared library for basic-math functionality.
28 add_library( basicmath SHARED )
29 target_link_libraries( basicmath PUBLIC basicmath_ObjLib )
30
31 # A shared library for advanced-math functionality.
32 add_library( extmath SHARED )
33 target_sources( extmath
34     PRIVATE "src/ExtendedMath.cpp" # Premium-content!
35 target_link_libraries( extmath PUBLIC basicmath_ObjLib )
36 # "Boost.Graph" is an additional dependency.
37 target_link_libraries( extmath PRIVATE Boost::graph )
38
39 # Create ALIAS targets.
40 add_library( MyCalc::basicmath ALIAS basicmath )
41 add_library( MyCalc::extmath ALIAS extmath )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    └── boost/
        └── CMakeLists.txt
            └── boost_outcome/
                └── CMakeLists.txt
        └── library/
            └── CMakeLists.txt
        └── include/
            ├── MathAPI.h
            └── Math.h
        └── src/
            ├── BasicMath.cpp
            ├── ExtendedMath.cpp
            └── HeavyMath.cpp
```

LINKING ALL TOGETHER INTO EXECUTABLES

FREECALCULATOR / PREMIUMCALCULATOR

LINKING ALL TOGETHER INTO EXECUTABLES

TRADITIONAL AND Modern CMAKE WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```

```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
└── external/
    ├── CMakeLists.txt
    ├── boost/
    │   └── CMakeLists.txt
    └── boost_outcome/
        └── CMakeLists.txt
└── library/
    ├── CMakeLists.txt
    ├── include/
    │   ├── MathAPI.h
    │   └── Math.h
    └── src/
        ├── BasicMath.cpp
        ├── ExtendedMath.cpp
        └── HeavyMath.cpp
```

LINKING ALL TOGETHER INTO EXECUTABLES

TRADITIONAL AND *Modern CMake* WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```



```
01 # ./app/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( CalculatorApp
06           VERSION 1.0
07           DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/dummy.cpp" )
11 target_sources( FreeCalculator PRIVATE "src/main.cpp" )
12 target_link_libraries( FreeCalculator
13                         PRIVATE MyCalc::basicmath
14                         Boost::program_options )
15
16 # Premium-calculator app (with advanced functionality)
17 add_executable( PremiumCalculator "src/dummy.cpp" )
18 target_sources( PremiumCalculator PRIVATE "src/main.cpp" )
19 target_link_libraries( PremiumCalculator
20                         PRIVATE MyCalc::extmath
21                         Boost::program_options )
22
23 # Possibly, compile key-file into premium-calculator, too.
24 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
25     target_sources( PremiumCalculator
26                     PRIVATE "src/key-file.cpp" )
27 endif ()
```

```
.
  +-- CMakeLists.txt
  +-- app/
  |   +-- CMakeLists.txt
  |   +-- src/
  |       +-- key-file.cpp
  |       +-- main.cpp
  +-- external/
  |   +-- CMakeLists.txt
  |   +-- boost/
  |       +-- CMakeLists.txt
  |   +-- boost_outcome/
  |       +-- CMakeLists.txt
  +-- library/
  |   +-- CMakeLists.txt
  |   +-- include/
  |       +-- MathAPI.h
  |       +-- Math.h
  +-- src/
      +-- BasicMath.cpp
      +-- ExtendedMath.cpp
      +-- HeavyMath.cpp
```

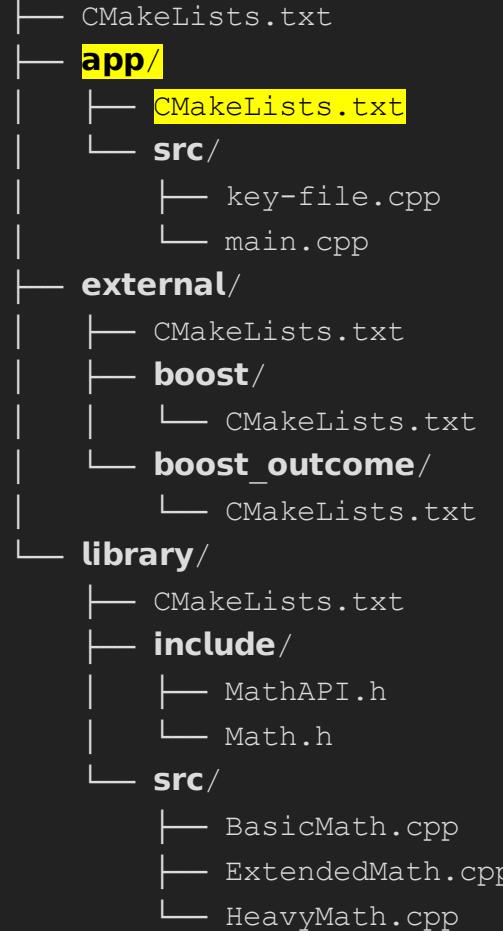
LINKING ALL TOGETHER INTO EXECUTABLES

TRADITIONAL AND *Modern CMake* WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```



```
01 # ./app/CMakeLists.txt -- Modern CMake
02
03 cmake_minimum_required( VERSION 3.10 )
04
05 project( CalculatorApp
06           VERSION 1.0
07           DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/dummy.cpp" )
11 target_sources( FreeCalculator PRIVATE "src/main.cpp" )
12 target_link_libraries( FreeCalculator
13                         PRIVATE MyCalc::basicmath
14                         Boost::program_options )
15
16 # Premium-calculator app (with advanced functionality)
17 add_executable( PremiumCalculator "src/dummy.cpp" )
18 target_sources( PremiumCalculator PRIVATE "src/main.cpp" )
19 target_link_libraries( PremiumCalculator
20                         PRIVATE MyCalc::extmath
21                         Boost::program_options )
22
23 # Possibly, compile key-file into premium-calculator, too.
24 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
25     target_sources( PremiumCalculator
26                     PRIVATE "src/key-file.cpp" )
27 endif ()
```



LINKING ALL TOGETHER INTO EXECUTABLES

TRADITIONAL AND More MODERN CMAKE WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```



```
01 # ./app/CMakeLists.txt -- More Modern CMake
02
03 cmake_minimum_required( VERSION 3.11 )
04
05 project( CalculatorApp
06           VERSION 1.0
07           DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator )
11 target_sources( FreeCalculator PRIVATE "src/main.cpp" )
12 target_link_libraries( FreeCalculator
13                         PRIVATE MyCalc::basicmath
14                         Boost::program_options )
15
16 # Premium-calculator app (with advanced functionality)
17 add_executable( PremiumCalculator )
18 target_sources( PremiumCalculator PRIVATE "src/main.cpp" )
19 target_link_libraries( PremiumCalculator
20                         PRIVATE MyCalc::extmath
21                         Boost::program_options )
22
23 # Possibly, compile key-file into premium-calculator, too.
24 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
25     target_sources( PremiumCalculator
26                     PRIVATE "src/key-file.cpp" )
27 endif ()
```

```
.
  +-- CMakeLists.txt
  +-- app/
  |   +-- CMakeLists.txt
  |   +-- src/
  |       +-- key-file.cpp
  |       +-- main.cpp
  +-- external/
  |   +-- CMakeLists.txt
  |   +-- boost/
  |       +-- CMakeLists.txt
  |   +-- boost_outcome/
  |       +-- CMakeLists.txt
  +-- library/
  |   +-- CMakeLists.txt
  |   +-- include/
  |       +-- MathAPI.h
  |       +-- Math.h
  +-- src/
      +-- BasicMath.cpp
      +-- ExtendedMath.cpp
      +-- HeavyMath.cpp
```

CMAKING AND COMPILING EVERYTHING

MODERN AND MORE MODERN CMAKE WAY

CMAKING

COMPILING

```
## cd /tmp/cmake/more_modern/build
## cmake -G "Unix Makefiles"../source/
-- The C compiler identification is GNU 6.5.0
-- The CXX compiler identification is GNU 6.5.0
-- Check for working C compiler: /usr/bin/gcc-6
-- Check for working C compiler: /usr/bin/gcc-6 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-6
-- Check for working CXX compiler: /usr/bin/g++-6 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   program_options
--   graph
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/cmake/more_modern/build
```

CMAKING AND COMPIILING EVERYTHING

MODERN AND MORE MODERN CMAKE WAY

CMAKING

SUCCESS!

COMPIILING

```
## cd /tmp/cmake/more_modern/build
## cmake -G "Unix Makefiles"../source/
-- The C compiler identification is GNU 6.5.0
-- The CXX compiler identification is GNU 6.5.0
-- Check for working C compiler: /usr/bin/gcc-6
-- Check for working C compiler: /usr/bin/gcc-6 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-6
-- Check for working CXX compiler: /usr/bin/g++-6 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   program_options
--   graph
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/cmake/more_modern/build
```

```
## cd /tmp/cmake/more_modern/build
## cmake --build .
Scanning dependencies of target basicmath_ObjLib
[ 10%] Building CXX object library/CMakeFiles/basicmath_ObjLib.dir/
src/BasicMath.cpp.o
[ 20%] Building CXX object library/CMakeFiles/basicmath_ObjLib.dir/
src/HeavyMath.cpp.o
[ 20%] Built target basicmath_ObjLib
Scanning dependencies of target extmath
[ 30%] Building CXX object library/CMakeFiles/extmath.dir/src/ExtendedMath.cpp.o
[ 40%] Linking CXX shared library libextmath.so
[ 40%] Built target extmath
Scanning dependencies of target basicmath
[ 50%] Linking CXX shared library libbasicmath.so
[ 50%] Built target basicmath
Scanning dependencies of target PremiumCalculator
[ 60%] Building CXX object app/CMakeFiles/PremiumCalculator.dir/src/main.cpp.o
[ 70%] Building CXX object app/CMakeFiles/PremiumCalculator.dir/src/key-file.cpp.o
[ 80%] Linking CXX shared library PremiumCalculator
[ 80%] Built target PremiumCalculator
Scanning dependencies of target FreeCalculator
[ 90%] Building CXX object app/CMakeFiles/FreeCalculator.dir/src/main.cpp.o
[100%] Linking CXX shared library FreeCalculator
[100%] Built target FreeCalculator
```

CMAKING AND COMPIILING EVERYTHING

TRADITIONAL CMAKE WAY

CMAKING

COMPIILING

```
## cd /tmp/cmake/traditional/build
## cmake -G "Unix Makefiles"../source/
-- The C compiler identification is GNU 6.5.0
-- The CXX compiler identification is GNU 6.5.0
-- Check for working C compiler: /usr/bin/gcc-6
-- Check for working C compiler: /usr/bin/gcc-6 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-6
-- Check for working CXX compiler: /usr/bin/g++-6 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   program_options
--   graph
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/cmake/traditional/build
```

CMAKING AND COMPIILING EVERYTHING

TRADITIONAL CMAKE WAY

CMAKING

FAILURE!

COMPIILING

```
## cd /tmp/cmake/traditional/build
## cmake -G "Unix Makefiles"../source/
-- The C compiler identification is GNU 6.5.0
-- The CXX compiler identification is GNU 6.5.0
-- Check for working C compiler: /usr/bin/gcc-6
-- Check for working C compiler: /usr/bin/gcc-6 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-6
-- Check for working CXX compiler: /usr/bin/g++-6 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   program_options
--   graph
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/cmake/traditional/build
```

```
## cd /tmp/cmake/traditional/build
## cmake --build .
Scanning dependencies of target basicmath_ObjLib
[ 10%] Building CXX object library/CMakeFiles/basicmath_ObjLib.dir/
src/BasicMath.cpp.o
[ 20%] Building CXX object library/CMakeFiles/basicmath_ObjLib.dir/
src/HeavyMath.cpp.o
[ 20%] Built target basicmath_ObjLib
Scanning dependencies of target extmath
[ 30%] Building CXX object library/CMakeFiles/extmath.dir/src/ExtendedMath.cpp.o
[ 40%] Linking CXX shared library libextmath.so
[ 40%] Built target extmath
Scanning dependencies of target basicmath
[ 50%] Linking CXX shared library libbasicmath.so
[ 50%] Built target basicmath
Scanning dependencies of target PremiumCalculator
[ 60%] Building CXX object app/CMakeFiles/PremiumCalculator.dir/src/main.cpp.o
/tmp/cmake/traditional/source/app/src/main.cpp:4:18: fatal error:
Math.h: No such file or directory
#include "Math.h"
^
compilation terminated.
...
```

LINKING ALL TOGETHER INTO EXECUTABLES

TRADITIONAL CMAKE WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```

```
.   CMakeLists.txt
+-- app/
|   CMakeLists.txt
|   src/
|       key-file.cpp
|       main.cpp
+-- external/
|   CMakeLists.txt
|   boost/
|       CMakeLists.txt
|   boost_outcome/
|       CMakeLists.txt
+-- library/
|   CMakeLists.txt
|   include/
|       MathAPI.h
|       Math.h
|   src/
|       BasicMath.cpp
|       ExtendedMath.cpp
|       HeavyMath.cpp
```

LINKING ALL TOGETHER INTO EXECUTABLES

FIXED TRADITIONAL CMAKE WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```

```
28 # Both targets require the include-path to their direct
29 # dependency 'basicmath'/'extmath' and indirect dependency
30 # "Boost.Outcome".
31 include_directories( "../library/include" ) # This is ugly!
32 include_directories( SYSTEM
33                     "${${TARGET_PROPERTY:Boost::outcome}_INCLUDE_DIRS}" )
```

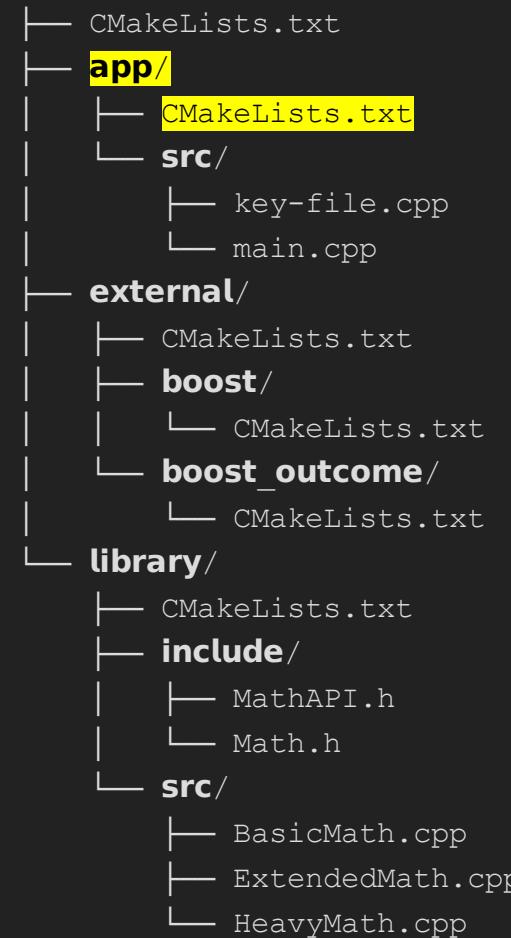
```
.
├── CMakeLists.txt
└── app/
    ├── CMakeLists.txt
    └── src/
        ├── key-file.cpp
        └── main.cpp
├── external/
│   ├── CMakeLists.txt
│   └── boost/
│       └── CMakeLists.txt
└── boost_outcome/
    └── CMakeLists.txt
library/
    ├── CMakeLists.txt
    └── include/
        ├── MathAPI.h
        └── Math.h
src/
    ├── BasicMath.cpp
    ├── ExtendedMath.cpp
    └── HeavyMath.cpp
```

LINKING ALL TOGETHER INTO EXECUTABLES

FIXED TRADITIONAL CMAKE WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```

```
28 # Both targets require the include-path to their direct
29 # dependency 'basicmath'/'extmath' and indirect dependency
30 # "Boost.Outcome".
31 include_directories( "../library/include" ) # This is ugly!
32 include_directories( SYSTEM
33                     "${TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS}" )
```



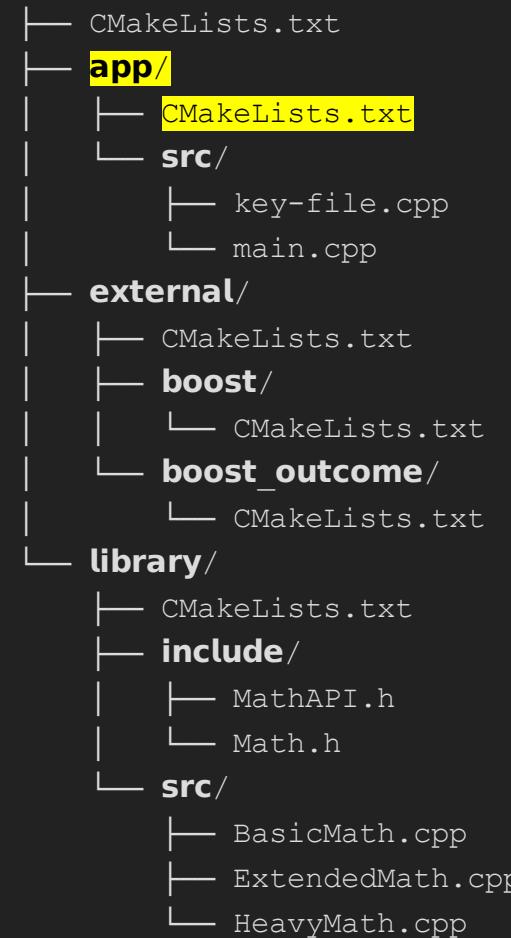
- *Modern CMake does not need this!*
 - Dependencies propagate their include-search-paths automatically, together with all other usage-requirements.

LINKING ALL TOGETHER INTO EXECUTABLES

FIXED TRADITIONAL CMAKE WAY

```
01 # ./app/CMakeLists.txt -- Traditional CMake
02
03 cmake_minimum_required( VERSION 2.8.10 )
04
05 project( CalculatorApp )
06 set( VERSION 1.0 )
07 set( DESCRIPTION "My fancy calculator app." )
08
09 # Freely available calculator app.
10 add_executable( FreeCalculator "src/main.cpp" )
11 target_link_libraries( FreeCalculator basicmath )
12
13 # Possibly, compile key-file into premium-calculator, too.
14 if (EXISTS "${CMAKE_CURRENT_SOURCE_DIR}/src/key-file.cpp")
15     set( EXTRA_SOURCES "src/key-file.cpp" )
16 endif ()
17
18 # Premium-calculator app (with advanced functionality)
19 add_executable( PremiumCalculator "src/main.cpp"
20                 ${EXTRA_SOURCES} )
21 target_link_libraries( PremiumCalculator extmath )
22
23 # Both targets require "Boost.Program_Options".
24 include_directories( SYSTEM "${Boost_INCLUDE_DIRS}" )
25 target_link_libraries( FreeCalculator ${Boost_LIBRARIES} )
26 target_link_libraries( PremiumCalculator
27                         ${Boost_LIBRARIES} )
```

```
28 # Both targets require the include-path to their direct
29 # dependency 'basicmath'/'extmath' and indirect dependency
30 # "Boost.Outcome".
31 include_directories( "../library/include" ) # This is ugly!
32 include_directories( SYSTEM
33                     "${TARGET_PROPERTY:Boost::outcome,MY_INCLUDE_DIRS}" )
```



- *Modern CMake does not need this!*
 - Dependencies propagate their include-search-paths automatically, together with all other usage-requirements.

LET'S CHECK...

CMAKING AND COMPIILING EVERYTHING

TRADITIONAL CMAKE WAY

CMAKING

COMPIILING

```
## cd /tmp/cmake/more_modern/build
## cmake -G "Unix Makefiles"../source/
-- The C compiler identification is GNU 6.5.0
-- The CXX compiler identification is GNU 6.5.0
-- Check for working C compiler: /usr/bin/gcc-6
-- Check for working C compiler: /usr/bin/gcc-6 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-6
-- Check for working CXX compiler: /usr/bin/g++-6 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   program_options
--   graph
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/cmake/more_modern/build
```

CMAKING AND COMPIILING EVERYTHING

TRADITIONAL CMAKE WAY

CMAKING

```
## cd /tmp/cmake/more_modern/build
## cmake -G "Unix Makefiles"../source/
-- The C compiler identification is GNU 6.5.0
-- The CXX compiler identification is GNU 6.5.0
-- Check for working C compiler: /usr/bin/gcc-6
-- Check for working C compiler: /usr/bin/gcc-6 -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/g++-6
-- Check for working CXX compiler: /usr/bin/g++-6 -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   program_options
--   graph
-- Configuring done
-- Generating done
-- Build files have been written to: /tmp/cmake/more_modern/build
```

FINALLY!

COMPIILING

```
## cd /tmp/cmake/more_modern/build
## cmake --build .
Scanning dependencies of target basicmath_ObjLib
[ 10%] Building CXX object library/CMakeFiles/basicmath_ObjLib.dir/
src/BasicMath.cpp.o
[ 20%] Building CXX object library/CMakeFiles/basicmath_ObjLib.dir/
src/HeavyMath.cpp.o
[ 20%] Built target basicmath_ObjLib
Scanning dependencies of target extmath
[ 30%] Building CXX object library/CMakeFiles/extmath.dir/src/ExtendedMath.cpp.o
[ 40%] Linking CXX shared library libextmath.so
[ 40%] Built target extmath
Scanning dependencies of target basicmath
[ 50%] Linking CXX shared library libbasicmath.so
[ 50%] Built target basicmath
Scanning dependencies of target PremiumCalculator
[ 60%] Building CXX object app/CMakeFiles/PremiumCalculator.dir/src/main.cpp.o
[ 70%] Building CXX object app/CMakeFiles/PremiumCalculator.dir/src/key-file.cpp.o
[ 80%] Linking CXX shared library PremiumCalculator
[ 80%] Built target PremiumCalculator
Scanning dependencies of target FreeCalculator
[ 90%] Building CXX object app/CMakeFiles/FreeCalculator.dir/src/main.cpp.o
[100%] Linking CXX shared library FreeCalculator
[100%] Built target FreeCalculator
```

TARGET — LINK — LIBRARIES LIBRARIES

SOME PECULIARITIES

THIS DOES NOT WORK.

```
01 # subdir/CMakeLists.txt
02 ...
03 add_library( someTarget SHARED )
04 target_sources( someTarget PRIVATE source1.cpp )
```

```
01 # CMakeLists.txt
02 ...
03 add_subdirectory( subdir )
04
05 add_library( someOtherTarget SHARED )
06 target_sources( someOtherTarget PRIVATE source2.cpp )
07
08 # Add a dependency to the build-requirements of a target from another directory.
09 target_link_libraries( someTarget PRIVATE someOtherTarget )
```

THIS DOES NOT WORK.

```
01 # subdir/CMakeLists.txt
02 ...
03 add_library( someTarget SHARED )
04 target_sources( someTarget PRIVATE source1.cpp )
```

```
01 # CMakeLists.txt
02 ...
03 add_subdirectory( subdir )
04
05 add_library( someOtherTarget SHARED )
06 target_sources( someOtherTarget PRIVATE source2.cpp )
07
08 # Add a dependency to the build-requirements of a target from another directory.
09 target_link_libraries( someTarget PRIVATE someOtherTarget )
```

- `target_link_libraries` called from *another directory scope*
 - cannot modify **build-requirements** of a target

HOWEVER, THIS DOES WORK.

```
01 # subdir/CMakeLists.txt
02 ...
03 add_library( someTarget SHARED )
04 target_sources( someTarget PRIVATE source1.cpp )
```

```
01 # CMakeLists.txt
02 ...
03 add_subdirectory( subdir )
04
05 add_library( someOtherTarget SHARED )
06 target_sources( someOtherTarget PRIVATE source2.cpp )
07
08 # Add a dependency to the usage-requirements of a target from another directory.
09 target_link_libraries( someTarget INTERFACE someOtherTarget )
```

- `target_link_libraries` called from *another directory scope*
 - cannot modify **build-requirements** of a target, but
 - can modify **usage-requirements** of a target

HOWEVER, THIS DOES WORK.

```
01 # subdir/CMakeLists.txt
02 ...
03 add_library( someTarget SHARED )
04 target_sources( someTarget PRIVATE source1.cpp )
```

```
01 # CMakeLists.txt
02 ...
03 add_subdirectory( subdir )
04
05 add_library( someOtherTarget SHARED )
06 target_sources( someOtherTarget PRIVATE source2.cpp )
07
08 # Add a dependency to the usage-requirements of a target from another directory.
09 target_link_libraries( someTarget INTERFACE someOtherTarget )
```

- `target_link_libraries` called from *another directory scope*
 - cannot modify **build-requirements** of a target, but
 - can modify **usage-requirements** of a target **by accident!**

CMAKE 3.13 LIFTS THIS RESTRICTION

- `target_link_libraries` can now be called from everywhere to add dependencies.
(<https://gitlab.kitware.com/cmake/cmake/issues/17943>)

CMAKE 3.13 LIFTS THIS RESTRICTION

- `target_link_libraries` can now be called from everywhere to add dependencies.
(<https://gitlab.kitware.com/cmake/cmake/issues/17943>)
- Additionally, CMake 3.13 now also allows to `install` targets created in different directory scope.
(<https://gitlab.kitware.com/cmake/cmake/issues/14444>)

SOME IDEAS FOR FUTURE CHANGES TO CMAKE

ALIASING UNKNOWN IMPORTED TARGETS

- Although ALIASing now also works with IMPORTED targets, it does not work completely.

ALIASING UNKNOWN IMPORTED TARGETS

- Although ALIASing now also works with IMPORTED targets, it does not work completely.
 - UNKNOWN IMPORTED targets still cannot be aliased.

ALIASING UNKNOWN IMPORTED TARGETS

- Although `ALIAS`ing now also works with `IMPORTED` targets, it does not work completely.
 - `UNKNOWN IMPORTED` targets still cannot be aliased.
- ⇒ This should be fixed.

<https://gitlab.kitware.com/cmake/cmake/issues/18327>

SIMPLIFY PROPAGATION OF *OBJECT-FILES*
AND
ALLOW CIRCULAR-DEPENDENCIES AMONG **OBJECT TARGETS**

- The propagation of *object-files* from **OBJECT** targets is weird (as you have seen).

SIMPLIFY PROPAGATION OF *OBJECT-FILES*
AND
ALLOW CIRCULAR-DEPENDENCIES AMONG **OBJECT TARGETS**

- The propagation of *object-files* from **OBJECT** targets is weird (as you have seen).
 - Especially, because propagation from one **OBJECT** target to another does not happen.

SIMPLIFY PROPAGATION OF *OBJECT-FILES*

AND

ALLOW CIRCULAR-DEPENDENCIES AMONG **OBJECT** TARGETS

- The propagation of *object-files* from **OBJECT** targets is weird (as you have seen).
 - Especially, because propagation from one **OBJECT** target to another does not happen.
- ⇒ This should become simpler.

<https://gitlab.kitware.com/cmake/cmake/issues/17905>
<https://gitlab.kitware.com/cmake/cmake/issues/18090>

ADD_CUSTOM_COMMAND AND OBJECT TARGETS

- `add_custom_command` can be used with `PRE_BUILD` or `POST_BUILD` option to call some commands *before* or *after* building a target.

ADD_CUSTOM_COMMAND AND OBJECT TARGETS

- `add_custom_command` can be used with `PRE_BUILD` or `POST_BUILD` option to call some commands *before* or *after* building a target.
 - However, this still does not work with `OBJECT` targets.

ADD_CUSTOM_COMMAND AND OBJECT TARGETS

- `add_custom_command` can be used with `PRE_BUILD` or `POST_BUILD` option to call some commands *before* or *after* building a target.
 - However, this still does not work with `OBJECT` targets.
⇒ This should be fixed.

<https://gitlab.kitware.com/cmake/cmake/issues/17081>

LET'S WRAP IT UP

TAKEAWAY

TAKEAWAY

- Switch to *Modern CMake!*

TAKEAWAY

- Switch to ~~Modern CMake!~~ More Modern CMake!

TAKEAWAY

- Switch to ~~Modern CMake!~~ *More Modern CMake!*
 - It is cleaner.
 - It is smaller.
 - It is less error-prone.

TAKEAWAY

- Switch to ~~Modern CMake!~~ **More Modern CMake!**
 - It is cleaner.
 - It is smaller.
 - It is less error-prone.
- Use newest CMake version if possible. (At least CMake 3.12.)

TAKEAWAY

- Switch to ~~Modern CMake!~~ **More Modern CMake!**
 - It is cleaner.
 - It is smaller.
 - It is less error-prone.
- Use newest CMake version if possible. (At least CMake 3.12.)
 - Always create targets with no sources, first.

TAKEAWAY

- Switch to ~~Modern CMake!~~ *More Modern CMake!*
 - It is cleaner.
 - It is smaller.
 - It is less error-prone.
- Use newest CMake version if possible. (At least CMake 3.12.)
 - Always create targets with no sources, first.
 - Use `target_...` commands to add *build-/usage-requirements*.

TAKEAWAY

- Switch to ~~Modern CMake!~~ **More Modern CMake!**
 - It is cleaner.
 - It is smaller.
 - It is less error-prone.
- Use newest CMake version if possible. (At least CMake 3.12.)
 - Always create targets with no sources, first.
 - Use `target_...` commands to add *build-/usage-requirements*.
 - Use `IMPORTED` targets for external libraries.
But, prefer `find_package` or `EXPORT`ed targets to creating them yourself.

TAKEAWAY

- Switch to ~~Modern CMake!~~ *More Modern CMake!*
 - It is cleaner.
 - It is smaller.
 - It is less error-prone.
- Use newest CMake version if possible. (At least CMake 3.12.)
 - Always create targets with no sources, first.
 - Use `target_...` commands to add *build-/usage-requirements*.
 - Use `IMPORTED` targets for external libraries.
But, prefer `find_package` or `EXPORT`ed targets to creating them yourself.
- Contribute to CMake!
 - Write issues in CMake's issue-tracker for errors and missing features.
 - Try to provide merge-requests for these (and other) issues.

REFERENCES

- *Craig Scott's "Professional CMake: A Practical Guide" e-book*
Buy it at: <https://crascit.com/professional-cmake/>
- The living document about *Modern CMake*
Read/Contribute at: <https://cliutils.gitlab.io/modern-cmake/>
- *Daniel Pfeiffer's "Effective CMake" talk*
Watch it at: <https://www.youtube.com/watch?v=bsXLMQ6WgIk>
- *Steven Kelly's "Embracing Modern CMake" talk*
Watch it at: <https://www.youtube.com/watch?v=Jsjl5xr1jxM>
- *Mathieu Ropert's "Modern CMake for modular design" talk*
Watch it at: <https://www.youtube.com/watch?v=ztrnb-bVVPo>

THANK YOU!

ANY QUESTIONS?

Examples from slides: <http://github.com/Bagira80/More-Modern-CMake>

BONUS SLIDES

BASICS FOR (MORE) MODERN CMAKE

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

- Call `cmake_minimum_required`

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

- Call `cmake_minimum_required`
 - *required*: at begin of **top-level** `CMakeLists.txt` file.

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

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

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

- 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.)

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

- 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.)
- **At least use version 3.12 as minimal version!**

CMAKE_MINIMUM_REQUIRED

```
01 # Minimal version 3.12, maximal version 3.13.  
02 cmake_minimum_required( VERSION 3.12...3.13 )
```

- 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.)
- **At least use version 3.12 as minimal version!**
 - The *version range* `<min-version>...<max-version>` syntax was introduced in 3.12, but is backwards-compatible.

PROJECT

```
01 # Define a project for the current CMakeLists.txt.  
02 project( <project_name>  
03           VERSION <major>[.<minor>[.<patch>[.<tweak>] ]]  
04           DESCRIPTION <project_description_string>  
05           [HOMEPAGE_URL <url_string>]  
06           [LANGUAGES <language_name>... ] )
```

PROJECT

```
01 # Define a project for the current CMakeLists.txt.  
02 project( <project_name>  
03           VERSION <major>[.<minor>[.<patch>[.<tweak>] ]]  
04           DESCRIPTION <project_description_string>  
05           [HOMEPAGE_URL <url_string>]  
06           [LANGUAGES <language_name>... ] )
```

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

PROJECT

```
01 # Define a project for the current CMakeLists.txt.  
02 project( <project_name>  
03           VERSION <major>[.<minor>[.<patch>[.<tweak>] ]]  
04           DESCRIPTION <project_description_string>  
05           [HOMEPAGE_URL <url_string>]  
06           [LANGUAGES <language_name>... ] )
```

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

PROJECT

```
01 # Define a project for the current CMakeLists.txt.  
02 project( <project_name>  
03           VERSION <major>[.<minor>[.<patch>[.<tweak>] ]]  
04           DESCRIPTION <project_description_string>  
05           [HOMEPAGE_URL <url_string>]  
06           [LANGUAGES <language_name>... ] )
```

- 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 (!) ...`

INCLUDE

```
01 # Verbatim copy the content of a file.
02 include( <file_path | module>
03           [OPTIONAL]
04           [RESULT_VARIABLE <variable_name>]
05           [NO_POLICY_SCOPE] )
```

INCLUDE

```
01 # Verbatim copy the content of a file.  
02 include( <file_path | module>  
03           [OPTIONAL]  
04           [RESULT_VARIABLE <variable_name>]  
05           [NO_POLICY_SCOPE] )
```

- Similar to `#include "file_path"` as used in C/C++.
 - Includes the content of a file / module in the current file.

INCLUDE

```
01 # Verbatim copy the content of a file.  
02 include( <file_path | module>  
03           [OPTIONAL]  
04           [RESULT_VARIABLE <variable_name>]  
05           [NO_POLICY_SCOPE] )
```

- Similar to `#include "file_path"` as used in C/C++.
 - Includes the content of a file / module in the current file.
- If not a file ⇒ a module `<module>.cmake`

INCLUDE

```
01 # Verbatim copy the content of a file.
02 include( <file_path | module>
03           [OPTIONAL]
04           [RESULT_VARIABLE <variable_name>]
05           [NO_POLICY_SCOPE] )
```

- Similar to `#include "file_path"` as used in C/C++.
 - Includes the content of a file / module in the current file.
- If not a file ⇒ a module `<module>.cmake`
 1. searched in paths from `CMAKE_MODULE_PATH` variable.
 2. searched in CMake's default module-search path.

INCLUDE

```
01 # Verbatim copy the content of a file.
02 include( <file_path | module>
03           [OPTIONAL]
04           [RESULT_VARIABLE <variable_name>]
05           [NO_POLICY_SCOPE] )
```

- Similar to `#include "file_path"` as used in C/C++.
 - Includes the content of a file / module in the current file.
- If not a file ⇒ a module `<module>.cmake`
 1. searched in paths from `CMAKE_MODULE_PATH` variable.
 2. searched in CMake's default module-search path.
- `RESULT_VARIABLE`: The full filename of the included file will be stored in `<variable_name>`.

INCLUDE

```
01 # Verbatim copy the content of a file.
02 include( <file_path | module>
03           [OPTIONAL]
04           [RESULT_VARIABLE <variable_name>]
05           [NO_POLICY_SCOPE] )
```

- Similar to `#include "file_path"` as used in C/C++.
 - Includes the content of a file / module in the current file.
- If not a file ⇒ a module `<module>.cmake`
 1. searched in paths from `CMAKE_MODULE_PATH` variable.
 2. searched in CMake's default module-search path.
- `RESULT_VARIABLE`: The full filename of the included file will be stored in `<variable_name>`.
- `OPTIONAL` and `NO_POLICY_SCOPE`: Mostly, not needed.

INCLUDE_GUARD

```
01 # Prevent including this file multiple times.  
02 include_guard( [DIRECTORY | GLOBAL] )
```

INCLUDE_GUARD

```
01 # Prevent including this file multiple times.  
02 include_guard( [DIRECTORY | GLOBAL] )
```

- Similar to `#pragma once` as used by many C++ compilers.

INCLUDE_GUARD

```
01 # Prevent including this file multiple times.  
02 include_guard( [DIRECTORY | GLOBAL] )
```

- Similar to `#pragma once` as used by many C++ compilers.
- Prevents including the current file again
 - `GLOBAL`: everywhere,
 - `DIRECTORY`: in this directory or its subdirectories,
 - `no args`: in variable scope.

EXAMPLE FOR CMAKELISTS . TXT BEGIN

```
01 # CMakeLists.txt
02
03 cmake_minimum_required( VERSION 3.12...3.13 )
04
05 # Get version and description from other file.
06 include( "${CMAKE_CURRENT_LIST_DIR}/project-meta-info.in" )
07 # Use version and description variables from other file.
08 project( MyExampleProject
09         VERSION ${project_version}
10         DESCRIPTION ${project_description} )
11 ...
```

```
01 # project-meta-info.in
02
03 include_guard()
04
05 # The version number of this project.
06 set( project_version 0.98.2 )
07 # The description of this project.
08 set( project_description "This is just an example project. "
09         "Do not expect anything here." )
```