I create this example at C:\Users\klieb\cmake\_testing

The short answer is, that each sub-directory has it's own variable scope initialized with a copy of the current variable values at the time of the add\_subdirectory() call.

For the long answer please see [What's the CMake syntax to set and use variables?](https://stackoverflow.com/questions/31037882/whats-the-cmake-syntax-to-set-and-use-variables)

**Directory & Target Properties vs (Global) Variables**

There is a difference between how [add\_compile\_options()](https://cmake.org/cmake/help/v3.4/command/add_compile_options.html) and [CMAKE\_CXX\_FLAGS](https://cmake.org/cmake/help/v3.4/variable/CMAKE_LANG_FLAGS.html) are processed by CMake:

* Everything that you specify with add\_compile\_options() is appended to the [COMPILE\_OPTIONS](https://cmake.org/cmake/help/v3.4/prop_dir/COMPILE_OPTIONS.html#prop_dir:COMPILE_OPTIONS) directory property. Then "this property is used to initialize the COMPILE\_OPTIONS target property when a target is created" with add\_library() or add\_executable().

And the current state of directory properties are used to initialize sub-directory properties when the parser gets to a add\_subdirectory() call.

* The [CMAKE\_CXX\_FLAGS](https://cmake.org/cmake/help/v3.4/variable/CMAKE_LANG_FLAGS.html) is a global cached variable. You can extend/overwrite it by defining a local directory scoped variable (hiding the globally cached one).

Those variable's context is copied into a sub-directories scope on add\_subdirectory() (propagating to sub-directories).

And CMake looks into its value at the *end* of the each CMakeLists.txt file and applies this to all targets in the same CMakeLists.txt (allowing late declarations, see also *Complete Formula* and *Test Code* below).

* So for CMake versions < 3.0 the equivalent to add\_compile\_options() was [add\_definitions()](https://cmake.org/cmake/help/v3.4/command/add_definitions.html). The functionality is still there, but it was strange to mix definitions with compile options. So add\_compile\_options() was invented.

**The complete Generator-Formula for Compiler Flags**

It's in CMake's code (see [cmCommonTargetGenerator::GetFlags()](https://github.com/Kitware/CMake/blob/master/Source/cmCommonTargetGenerator.cxx#L297), [cmLocalGenerator::AddCompileOptions()](https://github.com/Kitware/CMake/blob/master/Source/cmLocalGenerator.cxx#L1155) and [cmLocalGenerator::AddLanguageFlags()](https://github.com/Kitware/CMake/blob/master/Source/cmLocalGenerator.cxx#L1816)).

This example shows a DEBUG build configuration library without exports, not taking into account the feature-based flags or something like CMAKE\_CXX\_USE\_RESPONSE\_FILE\_FOR\_INCLUDES or CMAKE\_QUOTE\_INCLUDE\_PATHS:

|  |
| --- |
| CMAKE\_CXX\_FLAGS // as set at the end of target's CMakeLists.txt  + CMAKE\_CXX\_FLAGS\_DEBUG  + Include Directories // pefixed with CMAKE\_INCLUDE\_FLAG\_CXX/CMAKE\_INCLUDE\_SYSTEM\_FLAG\_CXX  (CMAKE\_INCLUDE\_CURRENT\_DIR) ?  + CMAKE\_CURRENT\_SOURCE\_DIR + CMAKE\_CURRENT\_BINARY\_DIR  + CMAKE\_CXX\_IMPLICIT\_INCLUDE\_DIRECTORIES  + Target[INCLUDE\_DIRECTORIES]  + DependingTargets[INTERFACE\_INCLUDE\_DIRECTORIES]  + Define Flags // compiler flags given with add\_definitions()  + Target[COMPILE\_FLAGS] // deprecated  - Filtered by CMAKE\_CXX\_FLAG\_REGEX  + Target[COMPILE\_OPTIONS]  + DependingTargets[INTERFACE\_COMPILE\_OPTIONS] |

**Test Code**

For a better understanding here is my code for testing the compiler options and the results I get:

*Note*: Normally I would use add\_definitions() and target\_compile\_definitions() instead of add\_compile\_options() and target\_compile\_options() to set compiler definitions, but to demonstrate the propagating of compiler options I (mis-)used -D flags.

**CMakeLists.txt**

|  |
| --- |
| cmake\_minimum\_required(VERSION 3.0)  project(CxxFlagsTest)  set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -DCXX\_FLAG")  add\_compile\_options("-DCOMPILE\_OPTION")  add\_subdirectory(lib)  file(WRITE main.cpp "int main() { return 0; }")  add\_executable(main main.cpp)  target\_link\_libraries(main lib)  target\_compile\_options(main PRIVATE "-DMAIN\_COMPILE\_OPTION")  set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -DLATE\_CXX\_FLAG")  get\_target\_property(\_main\_compile\_options main COMPILE\_OPTIONS)  message(STATUS "main COMPILE\_OPTIONS: ${\_main\_compile\_options}")  get\_directory\_property(\_root\_compile\_options COMPILE\_OPTIONS)  message(STATUS "root COMPILE\_OPTIONS: ${\_root\_compile\_options}")  message(STATUS "root CMAKE\_CXX\_FLAGS: ${CMAKE\_CXX\_FLAGS}") |

**lib/CMakeLists.txt**

|  |
| --- |
| set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -DSUB\_CXX\_FLAG")  add\_compile\_options("-DSUB\_COMPILE\_OPTION")  file(WRITE lib.cpp "")  add\_library(lib lib.cpp)  target\_compile\_options(lib PUBLIC "-DLIB\_COMPILE\_OPTION")  set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -DLATE\_SUB\_CXX\_FLAG")  get\_target\_property(\_lib\_compile\_options lib COMPILE\_OPTIONS)  message(STATUS "lib COMPILE\_OPTIONS: ${\_lib\_compile\_options}")  get\_directory\_property(\_sub\_compile\_options COMPILE\_OPTIONS)  message(STATUS "sub COMPILE\_OPTIONS: ${\_sub\_compile\_options}")  message(STATUS "sub CMAKE\_CXX\_FLAGS: ${CMAKE\_CXX\_FLAGS}") |

Would result in the following messages:

-- lib COMPILE\_OPTIONS: -DCOMPILE\_OPTION;-DSUB\_COMPILE\_OPTION;-DLIB\_COMPILE\_OPTION

-- sub COMPILE\_OPTIONS: -DCOMPILE\_OPTION;-DSUB\_COMPILE\_OPTION

-- sub CMAKE\_CXX\_FLAGS: ... -DCXX\_FLAG -DSUB\_CXX\_FLAG -DLATE\_SUB\_CXX\_FLAG

-- main COMPILE\_OPTIONS: -DCOMPILE\_OPTION;-DMAIN\_COMPILE\_OPTION

-- root COMPILE\_OPTIONS: -DCOMPILE\_OPTION

-- root CMAKE\_CXX\_FLAGS: ... -DCXX\_FLAG -DLATE\_CXX\_FLAG

And the following pre-processor definitions being set:

***lib***

CXX\_FLAG

SUB\_CXX\_FLAG

LATE\_SUB\_CXX\_FLAG

COMPILE\_OPTION

SUB\_COMPILE\_OPTION

LIB\_COMPILE\_OPTION

***main***

CXX\_FLAG

LATE\_CXX\_FLAG

COMPILE\_OPTION

MAIN\_COMPILE\_OPTION

LIB\_COMPILE\_OPTION

The interesting parts here are the LATE CXX flags and the LIB compile option propagated the the linked library.

**References**

* [cmake - Global linker flag setting (for all targets in directory)](https://stackoverflow.com/questions/32698580/cmake-global-linker-flag-setting-for-all-targets-in-directory)
* [What's the CMake syntax to set and use variables?](https://stackoverflow.com/questions/31037882/whats-the-cmake-syntax-to-set-and-use-variables)