



UNIVERSITÀ DEGLI STUDI DI PADOVA

Introducing CMake

Stefano Ghidoni





- Describing a software project
- Building systems
- CMake
 - Advantages
 - User's perspective
 - Developer's perspective



- A software project is usually composed of several source files (.h, .hpp, .cpp)
- Complex projects are composed of:
 - One or more executable(s)
 - One or more library/ies



- Source files alone do not provide a description of the project
- Need to know the structure of the SW project
 - Deeper understanding of the SW project
 - Info about how to compile



- A SW project can be described by means of its **targets**
- Compilation targets
 - Create an executable
 - Create a library
- In both cases
 - List of the .cpp files to be compiled together
 - List of the libraries to be linked



- How can we describe a project?
- Description based on **compile command**
 - Impractical
 - Hard to interpret
- Description based on **IDE project**
 - Convenient
 - Platform-dependent
- Description based on a **building system**



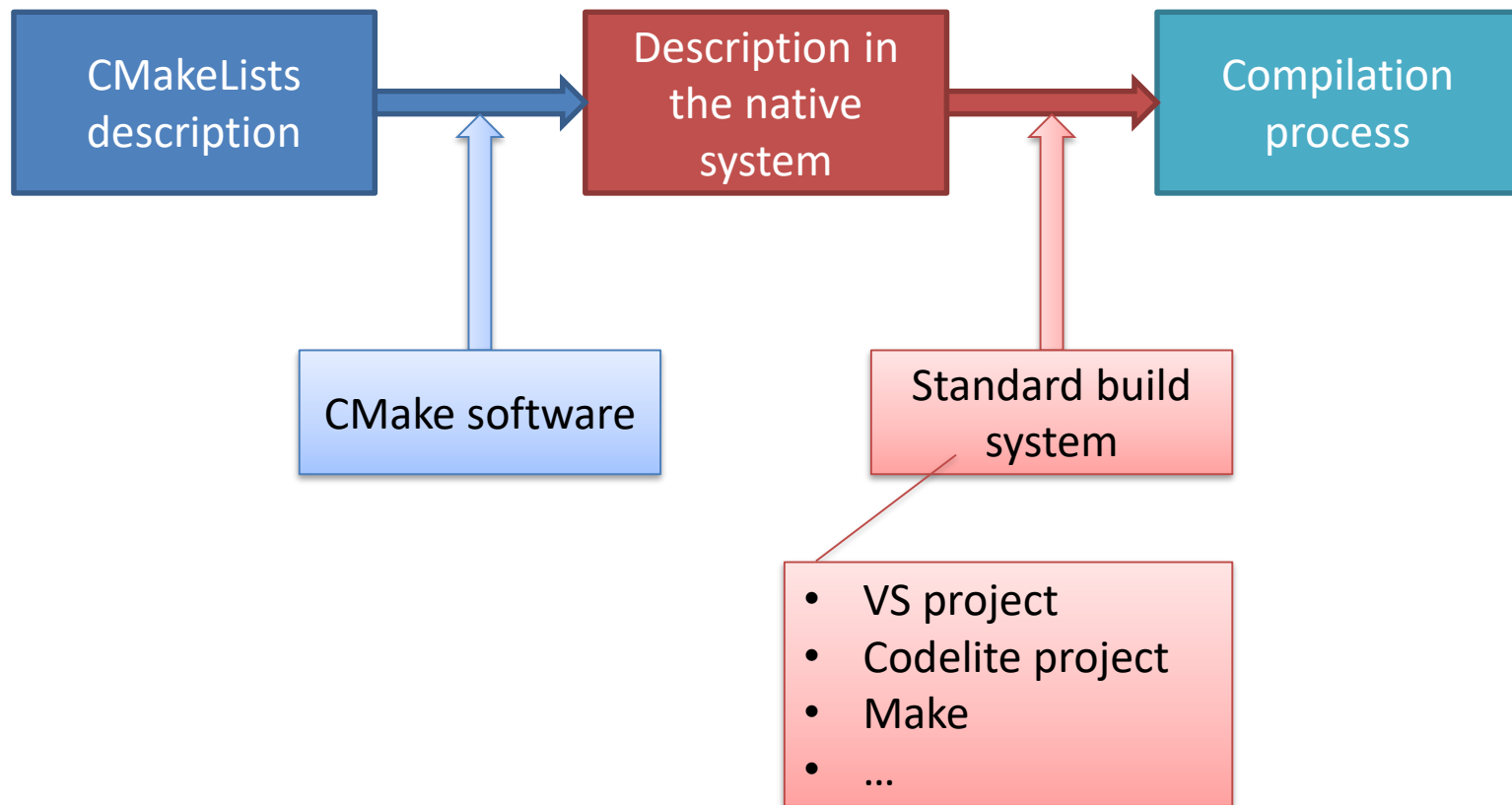
- A building system handles the compilation process
- Several options
 - Make, Ninja, ...
 - Sometimes depend on the platform



- Standard meta-building system
- Independent from:
 - Compiler
 - Platform
- Used in combination with the "local" building system
- Supported by many IDEs



- CMake can be seen as a way for describing projects
- It defines:
 - Compilation targets
 - Executables
 - Libraries
 - Libraries to be linked
- Based on text files named CMakeLists.txt





UNIVERSITÀ
DEGLI STUDI
DI PADOVA

CMake homepage

IAS-LAB

kitware

kitware.com contact us

 **CMake**

about ▼ solutions ▼ getting started documentation customize

Enter Keyword   **DOWNLOAD**

 **CMake**
BUILD YOUR WORLD

CMake: A Powerful Software Build System

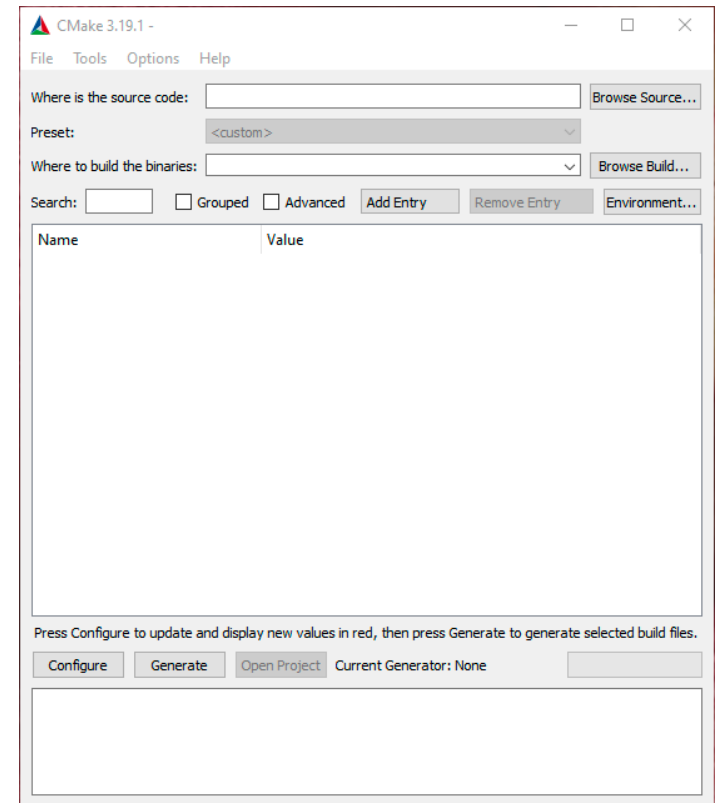
CMake is the de-facto standard for building C++ code, with over 2 million downloads a month. It's a powerful, comprehensive solution for managing the software build process. Get everything you need to successfully leverage CMake by visiting our resources section.

GETTING STARTED **DOCUMENTATION**

User's perspective



- GUI or command-line
- Two activities:
 - Configure
 - Generate





- Typical workflow:
 - **Call CMake** (GUI/command line)
 - Select the directory containing the source
 - Select the directory where the compiler output shall go
 - Standard: [project root]/build
 - **Configure**
 - (possible) module enable/disable
 - **Generate**



- The *generate* action creates the files to be provided to the build system
 - E.g.: VS solution
- From that point, the compilation process gets out of CMake's scope



- Example under Linux

```
mkdir build  
cd build  
cmake ..  
make
```

Compilation results in a separate folder (optional)



- Example under Linux

```
mkdir build  
cd build  
cmake ..  
make
```

- Generates compiler specific files
- **No compilation done yet!**



- Example under Linux

```
mkdir build  
cd build  
cmake ..  
make
```

- Compiles everything using a Makefile (generated by CMake)

Developer's perspective



- Describe the project using the CMake standard
 - Create CMakeLists.txt file manually
 - Use an IDE to generate CMakeLists.txt automatically
 - More complex
 - Do not modify them directly – only through the IDE



- Choose a minimum CMake version
 - 2.8 is usually enough

Sets the minimum required version of CMake.

```
1 cmake_minimum_required(VERSION 3.2 FATAL_ERROR)
```



- Give a name to the project

Set the name and version; enable languages.

```
1 project(<name> VERSION <version> LANGUAGES CXX)
```

- CMake sets several variables based on project().
- Call to project() must be direct, not through a function/macro/include.
- CMake will add a call to project() if not found on the top level.



- List libraries needed by the project
 - E.g., OpenCV

Finds preinstalled dependencies

```
1 find_package(Qt5 REQUIRED COMPONENTS Widgets)
```

- Can set some variables and define imported targets.
- Supports components.



- Target: creating an executable
 - Corresponds to: `g++ -o tool -c main.cpp another_file.cpp`

Add an executable target.

```
1  add_executable(tool
2      main.cpp
3      another_file.cpp
4  )
```




- Target: creating a library

Add a library target.

```
1  add_library(foo STATIC
2      foo1.cpp
3      foo2.cpp
4  )
```

- Libraries can be **STATIC**, **SHARED**, **MODULE**, or **INTERFACE**.
- Default can be controlled with **BUILD_SHARED_LIBS**.




```
1  cmake_minimum_required(VERSION 3.2 FATAL_ERROR)
2
3  project (tuple)
4
5  include_directories(include)
6
7  add_library(tuple
8      include/tuple.h
9      src/tuple.cpp
10 )
11
12 add_executable(add_tuple
13     src/add_tuple.cpp
14 )
15 target_link_libraries(add_tuple
16     tuple
17 )
```



```
1 cmake_minimum_required(VERSION 2.8)
2 project(test1)
3
4 find_package(OpenCV REQUIRED)
5
6 include_directories(${OpenCV_INCLUDE_DIRS})
7
8 add_executable(${PROJECT_NAME} src/test1.cpp)
9 target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```



```
1 cmake_minimum_required(VERSION 2.8)
2 project(test1)
3
4 find_package(OpenCV REQUIRED)
5
6 include_directories(${OpenCV_INCLUDE_DIRS})
7
8 add_executable(${PROJECT_NAME} src/test1.cpp)
9 target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```




Looks for
OpenCVConfig.cmake
(where a bunch of
OpenCV related vars
are defined)



```
1 cmake_minimum_required(VERSION 2.8)
2 project(test1)
3
4 find_package(OpenCV REQUIRED)
5
6 include_directories(${OpenCV_INCLUDE_DIRS})
7
8 add_executable(${PROJECT_NAME} src/test1.cpp)
9 target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```

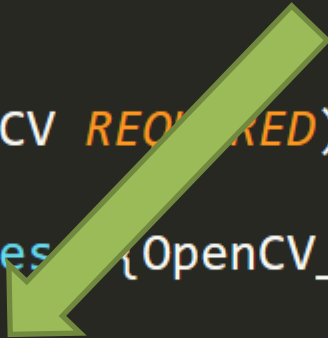
Tells the compiler to
look for the headers
also in the folders
defined in this variable





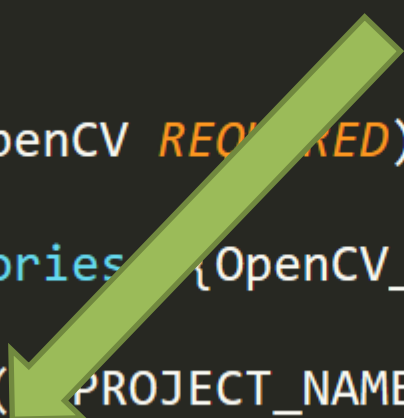
```
1 cmake_minimum_required(VERSION 2.8)
2 project(test1)
3
4 find_package(OpenCV REQUIRED)
5
6 include_directories(${OpenCV_INCLUDE_DIRS})
7
8 add_executable(${PROJECT_NAME} src/test1.cpp)
9 target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```

Tells the compiler to build an executable named test1, the source file for the compilation is just src/test1.cpp





```
1 cmake_minimum_required(VERSION 2.8)
2 project(test1)
3                                     Tells the linker to to link the test1
4 find_package(OpenCV REQUIRED) by the variable
5                                     executable with the libraries defined
6 include_directories(${OpenCV_INCLUDE_DIRS})
7
8 add_executable(${PROJECT_NAME} src/test1.cpp)
9 target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```





UNIVERSITÀ DEGLI STUDI DI PADOVA

Introducing CMake

Stefano Ghidoni

