

# **UNDERSTANDING CMAKE**

**P V S PHANEENDRA**

# AGENDA

- Overview on Build Process
- Build Files / Scripts
- Build Systems
- CMake - The Quintessential Build Tools Generator
- Getting Started with CMake
- CMake Language Constructs Overview
- Few Important CMake Commands
- Few Important CMake Variables
- Illustrations By Examples
- And More In Future
- Interesting Reads or Links
- References

# OVERVIEW ON BUILD PROCESS

- Why does a programmer require a build system ?
  - To execute one's code on a hardware
  - And, hardware understands **ONLY machine code**
  - And, programmers generally would be coding at a much higher abstraction level (C, C++, Python, etc.)
  - Now, how does the programmer **transform** the source at higher abstraction level to the machine code (host code) ?

```
volatile uint32_t * ptr = (volatile uint32_t *) (0x1001ffff);  
*ptr = 0xabcdef12;
```



```
101010100001010101010110  
101010101010101010001000
```

# OVERVIEW ON BUILD PROCESS

## [CONTD.. 1]

- Build process is tightly coupled with the *language design* and *decisions of the language committee*
- No one rule for all languages
- Each language has it's own build process in place for transforming the high level language to the machine code
- Build process consists of varying number of build steps for each language
- Hence, the need for understanding the build process arises !

# OVERVIEW ON BUILD PROCESS (CONTD.. 2)

- Classification of languages
  - based on abstraction levels closer to machine code
    - Low level languages (C, C++)
    - Interpreter languages (JavaScript, Python, etc.)
    - Virtual machine based languages (Java - JVM)
- Why do you think above classification is important and relevant in the current discussion ?
- Heard about *pre-processor*, *compiler*, *assembler* and *linker* ?

# BUILD FILES / SCRIPTS

- Build files usually are written to build a given source code for limited number of
  - Compilers
  - Platforms
- Few well known scripts : Makefile, npm scripts, runjs
- Executing Makefile
  - Build commands '**make**' (Unix flavours) or '**nmake**' (Win) use these build scripts in order to build the final executable
- Build scripts, for instance, contain list of **dependencies** and **rules**
- The rules within a build script contains
  - rules that determine the order in which the intermittent targets are built, finally resolving into generation of the final target file, and
  - the correct sequence of the rules

# BUILD FILES / SCRIPTS (CONTD.. 1)

```
package \  
|__ include \  
|    abcInteface.h, abcDetails.h  
|__ src \  
|    abcInterface.cpp, abcDetails.cpp
```

## Sample #1

```
all : g++ -o exec main.cpp -I${CWD_PATH}/package/include  
-L${CWD_PATH}/package/lib -lpackage  
  
clean :  
    @rm exec
```

## Sample #2

```
CXX = g++  
CFLAGS = -g -Wall -static  
AR = ar  
  
PKG_NAME := package  
INC_DIR := -I${CWD_PATH}/${PKG_NAME}/include  
SRC_DIR := ${CWD_PATH}/${PKG_NAME}/src  
OBJ_DIR := ${CWD_PATH}/${PKG_NAME}/obj  
  
SRCS := $(foreach s_dir, $(SRC_DIR), $(wildcard ${s_dir}/*.cpp))  
OBJS := $(patsubst ${SRC_DIR}/%.cpp, ${OBJ_DIR}/%.o, ${SRCS})  
  
LIBRARY := $(addprefix lib, ${PKG_NAME})  
LIBRARY := $(addsuffix .a, ${LIBRARY})  
LIBRARY := $(addprefix lib/, ${LIBRARY})  
  
vpath %.cpp $(SRC_DIR)  
vpath %.o ${OBJ_DIR}  
  
all : createDir ${LIBRARY}  
    @echo  
    @echo "Creating ${PKG_NAME} library ..."  
  
-include (OBJS:.o=.d)  
  
createDir :  
    @echo  
    @echo "Creating 'obj', 'lib' - required directories ..."  
    @mkdir -p obj  
    @mkdir -p lib  
  
define make-object  
$1/%.o : %.cpp  
    @echo "... " ${CXX} -MMD -MP $(CFLAGS) $(INC_DIR) -c $$< -o $$@  
endef  
  
${LIBRARY} : (OBJS)  
    @echo  
    $(AR) -r $$@ $^  
  
$(foreach b_dir, ${OBJ_DIR}, $(eval $(call make-object, $(b_dir))))  
  
clean :  
    @echo  
    @echo "Cleaning ${PKG_NAME} ..."  
    @rm ${OBJ_DIR} -rf  
    @rm lib -rf
```

# BUILD SYSTEMS

- Build Systems are far more generic than to build Scripts
  - Few of these tools have their own language
  - Far more flexible than build files alone
  - User scripting becomes easier (Eg. User need not remember more obscure make variables like \$@, \$^, \$+, etc.)
  - Build systems could be the build script generators as well
- Popular Build Tools
    - SCons (Sconstruct)
    - GNU Autotools
    - CMake
    - Jam
    - qmake
    - Ant
    - Maven
    - Gradle
    - Rake
    - Makepp



# CMAKE – THE QUINTESSENTIAL BUILD TOOLS GENERATOR

## CMake supports

- Command-line build tool generators
- IDE build tool generators
- Extra build tool generators

## • Companies / Users

- Netflix
- The HDF Group
- Inria
- Biicode
- ReactOS
- KDE
- Apache QPid
- Second Life

and many other prominent users ...

# CMAKE – THE QUINTESSENTIAL BUILD TOOLS GENERATOR (CONTD.. 1)

- **Command-line Generators**

- Borland Makefiles
- MSYS Makefiles
- MinGW Makefiles
- NMake Makefiles
- NMake Makefiles JOM
- Ninja
- Unix Makefiles
- Watcom Wmake

- **IDE build tool generators**

- Visual Studio xx 20xx
- Xcode

- **Extra build tool generators**

- CodeBlocks
- CodeLite
- Eclipse CDT4
- KDevelop3
- Kate
- Sublime Text 2

# SAMPLE ILLUSTRATION

- Session targeted towards C++ source code

## 00\_BASIC\_COMMAND\_LINE

▸ .vscode

M CMakeLists.txt

G+ command\_line.cpp

G+ command\_line.cpp ▸ ...

```
1  #include <iostream>
2
3  int main (int argc, char * argv[])
4  {
5      std::cout << "\nSimple command line example ..." << std::endl;
6      return 0;
7  }
```

M CMakeLists.txt

```
1  cmake_minimum_version (VERSION 3.0)
2
3  project (SimpleCommandLine)
4
5  set (SOURCES command_line.cpp)
6
7  add_executable(${PROJECT_NAME} ${SOURCES})
```

# SAMPLE ILLUSTRATION (CONTD.. 1)

Create a 'build' directory within the project and change current directory to that

The below snapshot shows how to create the solution and generate the relevant build files using CMake

```
PS C:\root\03_works\08_myRepos\learn-cmake\cmake_for_cpp\00_basic_command_line\build> cmake -G "Visual Studio 15 2017" ../
-- The C compiler identification is MSVC 19.16.27027.1
-- The CXX compiler identification is MSVC 19.16.27027.1
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2017/Professional/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x86/cl.exe
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2017/Professional/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x86/cl.exe -- work
s
-- 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: C:/Program Files (x86)/Microsoft Visual Studio/2017/Professional/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x86/cl.exe
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2017/Professional/VC/Tools/MSVC/14.16.27023/bin/Hostx86/x86/cl.exe -- wo
rks
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: C:/root/03_works/08_myRepos/learn-cmake\cmake_for_cpp\00_basic_command_line/build
PS C:\root\03_works\08_myRepos\learn-cmake\cmake_for_cpp\00_basic_command_line\build> |
```

# SAMPLE ILLUSTRATION (CONTD.. 2)

## BUILD DIRECTORY CONTENTS

```
Length Name
-----
      CMakeFiles
43783 ALL_BUILD.vcxproj
  330 ALL_BUILD.vcxproj.filters
13859 CMakeCache.txt
  1544 cmake_install.cmake
  3217 SimpleCommandLine.sln
53983 SimpleCommandLine.vcxproj
  671 SimpleCommandLine.vcxproj.filters
42745 ZERO_CHECK.vcxproj
  573 ZERO_CHECK.vcxproj.filters
```

## CMAKEFILES CONTENTS

```
Length Name
-----
      1bcb469471806e80838ac16f9c7d1fdc
      3.14.0
      CMakeTmp
      86 cmake.check_cache
16538 CMakeOutput.log
38400 feature_tests.bin
  287 feature_tests.c
  5194 feature_tests.cxx
   55 generate.stamp
7992 generate.stamp.depend
  109 generate.stamp.list
  333 TargetDirectories.txt
```



createSolution.txt (Command Line)

# SAMPLE ILLUSTRATION (CONTD.. 3)

Next step is to build the executable

```
PS C:\root\03_works\08_myRepos\learn-cmake\cmake_for_cpp\00_basic_command_line\build> cmake --build . --config -- /m
Microsoft (R) Build Engine version 15.9.21+g9802d43bc3 for .NET Framework
Copyright (C) Microsoft Corporation. All rights reserved.

Checking Build System
CMake does not need to re-run because C:/root/03_works/08_myRepos/learn-cmake/cmake_for_cpp/00_basic_command_line/build/CMakeFiles/generate.stamp is up
-to-date.
Building Custom Rule C:/root/03_works/08_myRepos/learn-cmake/cmake_for_cpp/00_basic_command_line/CMakeLists.txt
CMake does not need to re-run because C:/root/03_works/08_myRepos/learn-cmake/cmake_for_cpp/00_basic_command_line/build/CMakeFiles/generate.stamp is up
-to-date.
command_line.cpp
SimpleCommandLine.vcxproj -> C:\root\03_works\08_myRepos\learn-cmake\cmake_for_cpp\00_basic_command_line\build\Debug\SimpleCommandLine.exe
Building Custom Rule C:/root/03_works/08_myRepos/learn-cmake/cmake_for_cpp/00_basic_command_line/CMakeLists.txt
CMake does not need to re-run because C:/root/03_works/08_myRepos/learn-cmake/cmake_for_cpp/00_basic_command_line/build/CMakeFiles/generate.stamp is up
-to-date.
```

This would lead to generation of the object files and building of the final executable



buildTreeView.txt

# SAMPLE ILLUSTRATION (CONTD.. 4)

Traverse to the directory listing the final executable and let's run it

```
PS C:\root\03_works\08_myRepos\learn-cmake\cmake_for_cpp\00_basic_command_line\build> .\Debug\SimpleCommandLine.exe  
Simple command line example ...  
PS C:\root\03_works\08_myRepos\learn-cmake\cmake_for_cpp\00_basic_command_line\build> |
```

# CMAKE LANGUAGE CONSTRUCTS OVERVIEW

- CMake language construct can be classified into the following
  - CMake Commands
  - CMake Variables
- CMake (cmake.exe) Commands are necessary
  - CMake is a scripting language, and
  - It understands its commands ONLY
- CMake variables are necessary since they help in locating paths, positioning of the to-be created artefacts (build directories, binary directories, etc.), etcetera.



# FEW IMPORTANT CMAKE COMMANDS

## Basic Commands

cmake\_minimum\_required

project

set

message

add\_executable

## Working With Libraries

include\_directories

target\_include\_directories

link\_library

target\_link\_libraries

## Multi-Project Workspace

add\_subdirectory

set\_project\_properties

add\_dependencies

Adding Targets

- add\_library
- add\_executable

## Playing with Macros

target\_compile\_definitions

target\_compile\_options

## Find / locate

find\_file

find\_path

find\_package

find\_library

find\_program

## Configurations & Customizations

configure\_file

configure\_command

add\_custom\_command

add\_custom\_target

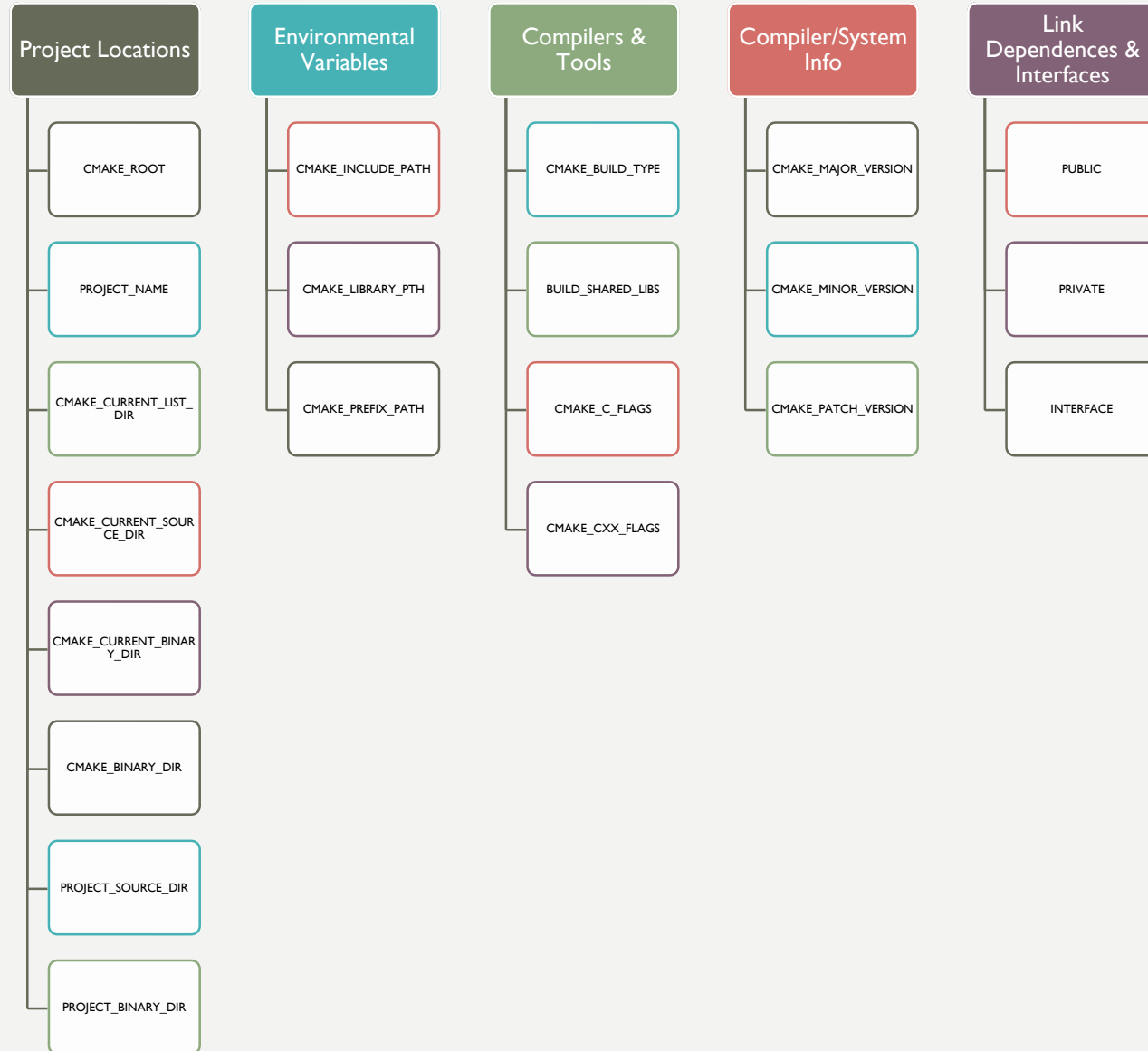
## Testing

Include(CTest)

add\_test

enable\_testing

# FEW IMPORTANT CMAKE VARIABLES



# ILLUSTRATIONS BY EXAMPLES

- Few often used CPP project templates have been created that illustrate some recommended ways of using CMake listfiles
- They range from:
  - Simple project creation (say, Hello-World program)
  - Adding multi-project workspace
  - Creating Libraries (static and dynamic)
  - Using NuGet packages

These are just to name a few

Modern-CMake rules/recommendations are not yet fully demonstrated

# AND MORE IN FUTURE

- Setting up properties for C++ project properties that enable strict language checking
- Demonstrating different build types supported by CMake
- More modern way of exporting DLL interfaces and consuming them
- Importance of CMake functions and their usage
- Using CMake – the ‘**config**’ way
- Demonstrating CMake integration with Conan

# INTERESTING READS AND LINKS

- <https://softwareengineering.stackexchange.com/questions/297847/why-do-build-tools-use-a-scripting-language-different-than-underlying-programmin>
- [https://www.cs.virginia.edu/~dww4s/articles/build\\_systems.html](https://www.cs.virginia.edu/~dww4s/articles/build_systems.html)
- <https://stackoverflow.com/questions/3209517/why-should-one-use-a-build-system-over-that-which-is-included-as-part-of-an-ide>

# REFERENCES

- <https://cmake.org/documentation/>
- <https://cmake.org/cmake-tutorial/>
- <https://linux.die.net/man/1/cmakecommands>
- <https://gitlab.kitware.com/cmake/community/wikis/doc/cmake/Useful-Variables>



THANK YOU