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 *)(0x1001fffff);
*ptr = 0xabcdef12;
```

```
10101010000101010101010110
10101010101010101010001000
```

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)

RegModel \

|__ include \

| abc.h, abcCore.h

|__ src \

| abc.cpp, abcCore.cpp

Sample #1

```
all : g++ -o exec main.cpp -I${REG_INFRA}/RegModel/include
-L${REG_INFRA}/RegModel/lib –lRegModel

clean :
    @rm exec
```

Sample #2

```
CXX = g++
CFLAGS = -g -Wall –static
AR = ar

MODEL_NAME := RegModel
INC_DIR := -I${REG_INFRA}/$(MODEL_NAME)/include
SRC_DIR := ${REG_INFRA}/$(MODEL_NAME)/src
OBJ_DIR := ${REG_INFRA}/$(MODEL_NAME)/obj

SRCS := $(foreach s_dir, $(SRC_DIR), $(wildcard $(s_dir)/*.cpp))
OBJS := $(patsubst $(SRC_DIR)/%.cpp, $(OBJ_DIR)/%.o, $(SRCS))

LIBRARY := $(addprefix lib, $(MODEL_NAME))
LIBRARY := $(addsuffix .a, $(LIBRARY))
LIBRARY := $(addprefix lib/, $(LIBRARY))

vpath %.cpp $(SRC_DIR)
vpath %.o $(OBJ_DIR)

all : create_directories $(LIBRARY)
    @echo
    @echo "Creating $(MODEL_NAME) library ..."

-include (OBJS:.o=.d)

create_directories :
    @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 $$< -o $$@
    @echo `echo $$@ | sed "s/\/\\\//g"`
endef

$(LIBRARY) : $(OBJS)
    @echo
    @echo "Archiving object files to generate the STATIC library ... $(OBJS)"
     $(AR) -r $@ $^

$(foreach b_dir, $(OBJ_DIR), $(eval $(call make-object, $(b_dir))))

clean :
    @echo
    @echo "Cleaning $(MODEL_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

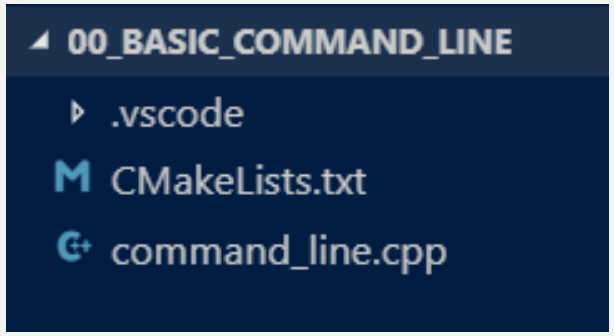
  and many other prominent users …

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

GETTING STARTED WITH CMAKE

- Session targeted towards C++ source code



command_line.cpp ▸ ...

```cpp
#include <iostream>

int main (int argc, char * argv[])
{
    std::cout << "\nSimple command line example ..." << std::endl;
    return 0;
}
```

CMakeLists.txt

```
cmake_minimim_version (VERSION 3.0)

project (SimpleCommandLine)

set (SOURCES command_line.cpp)

add_executable(${PROJECT_NAME} ${SOURCES})
```

GETTING STARTED WITH CMAKE (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>
```

GETTING STARTED WITH CMAKE (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)

GETTING STARTED WITH CMAKE (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

GETTING STARTED WITH CMAKE (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 it's commands ONLY
- CMake variables are necessary since they help in locating paths, positioning of the to-be created artefacts (build directories, binary directories, etc.), directing

FEW IMPORTANT CMAKE COMMANDS

- **Basic Commands**
  - cmake_minimum_required
  - project
  - set
  - message
  - add_executable
- **Customizations**
  - add_custom_command
  - add_custom_target
- **Working with libraries**
  - include_directories
  - target_include_directories
  - find_library
  - target_link_libraries
- **Testing**
  - Include(Ctest)
  - add_test
  - enable_testing
- **Find / locate**
  - find_file
  - find_package
- **Configurations**
  - configure_file
- **adding projects**
  - add_subdirectory

FEW IMPORTANT CMAKE VARIABLES

- **Locations**
  - CMAKE_ROOT
  - PROJECT_NAME
  - CMAKE_CURRENT_LIST_DIR
  - CMAKE_CURRENT_SOURCE_DIR
  - CMAKE_CURRENT_BINARY_DIR
  - CMAKE_BINARY_DIR
  - PROJECT_SOURCE_DIR
  - PROJECT_BINARY_DIR
- **Environmental Variables**
  - CMAKE_INCLUDE_PATH
  - CMAKE_LIBRARY_PTH
  - CMAKE_PREFIX_PATH
- **Compilers & Tools**
  - CMAKE_BUILD_TYPE
  - BUILD_SHARED_LIBS
  - CMAKE_C_FLAGS
  - CMAKE_CXX_FLAGS
- **Compiler/System Info**
  - CMAKE_MAJOR_VERSION
  - CMAKE_MINOR_VERSION
  - CMAKE_PATCH_VERSION

ILLUSTRATIONS BY EXAMPLES

- TBD

AND MORE IN FUTURE

- TBD

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://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