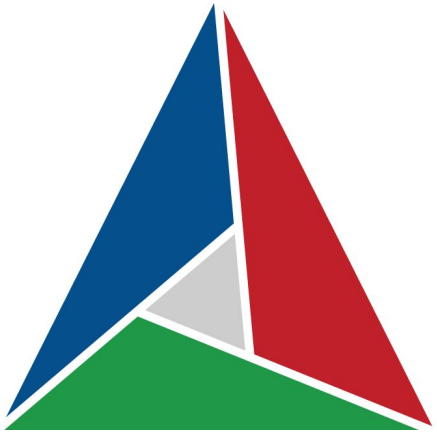


CMake

Crash Course on CMake

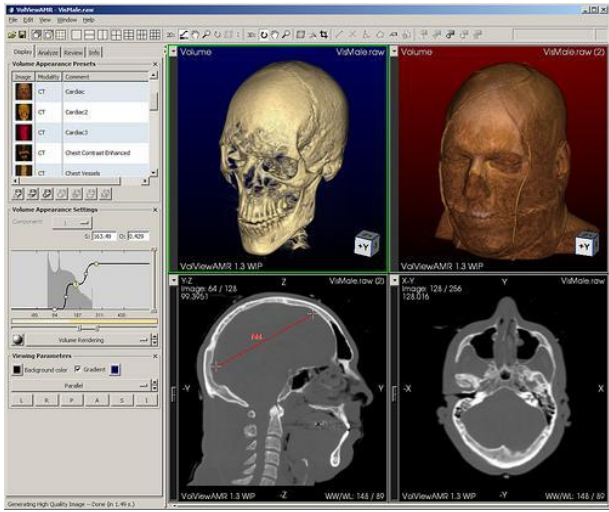
Robert Maynard, Principal Engineer @ NVIDIA

Where did CMake come from?

Kitware was the lead engineering team for the Insight Segmentation and Registration Toolkit (ITK) http://www.itk.org

Funded by National Library of Medicine (NLM): part of the Visible Human Project

CMake 1.0 released August 2001



What is CMake exactly?

Family of Software Development Tools

Build = CMake Test = CTest/CDash Package = CPack

CMake is the cross-platform, open-source build system that takes plain text files as input that describe your project and produces project files, make files, or ninja files

Consider CMake to replace autoconf not make

What is CMake exactly?

CMake design is to always generate correct incremental parallel build files

‘make -jN’ is always safe and correct

Only rebuild the subset of files / targets that have been modified

Will automatically re-run cmake on rebuild if any CMake file has changed

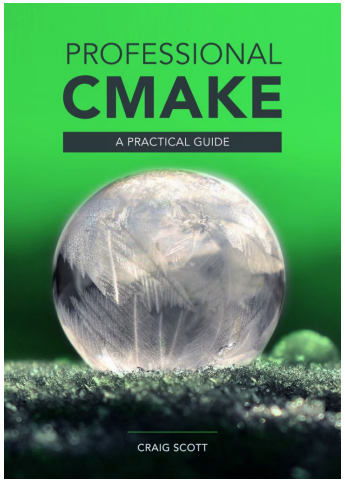
CMake Resources

The CMake reference documentation has a collection of tutorials both on writing CMake code, and using CMake

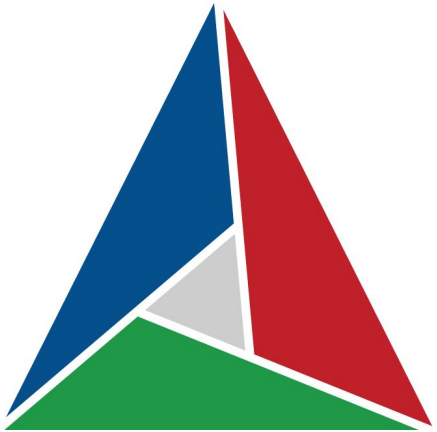
Guides

- CMake Tutorial
- User Interaction Guide
- Using Dependencies Guide
- Importing and Exporting Guide
- IDE Integration Guide

Professional CMake ebook by Craig Scott



discourse.cmake.org

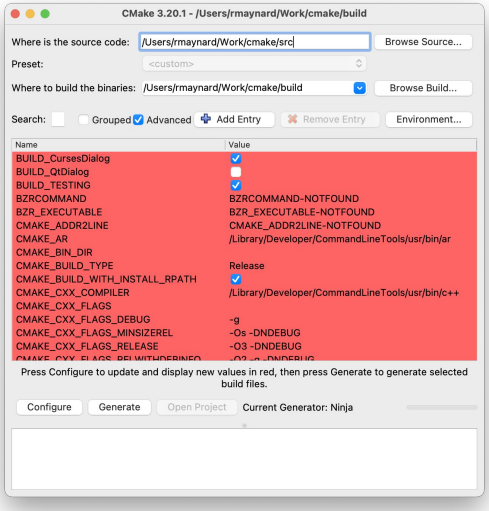


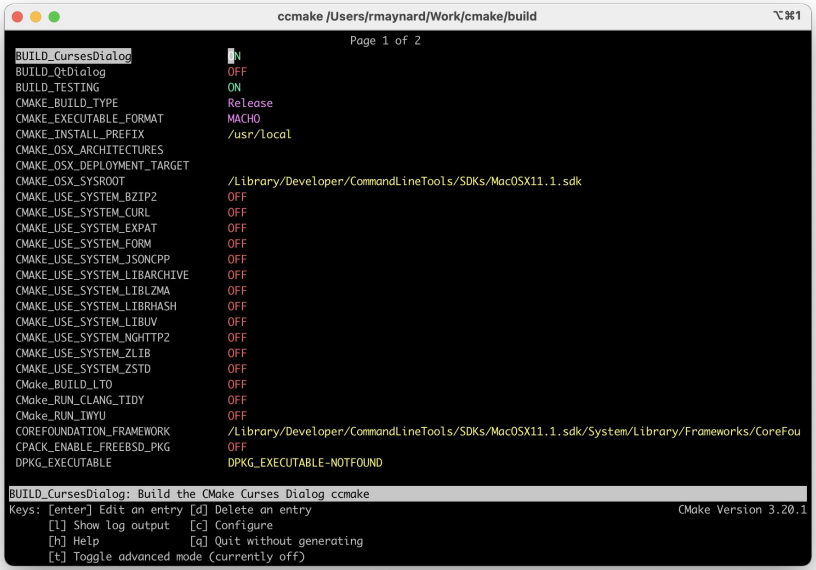
CMake

Running CMake

Running CMake

cmake -S <source_dir> -B <build_dir>
ccmake -S <source_dir> -B <build_dir>
cmake-gui





CMake Command Line Arguments

Common Arguments:

-S <source_directory>
-B <build_directory>
-G <generator_name>
-D<variable_name>:<value>
--install-prefix <directory> [3.21] (`-DCMAKE_INSTALL_PREFIX`)
--toolchain <path/to/file> [3.21] (`-DCMAKE_TOOLCHAIN_FILE`)

Modes:

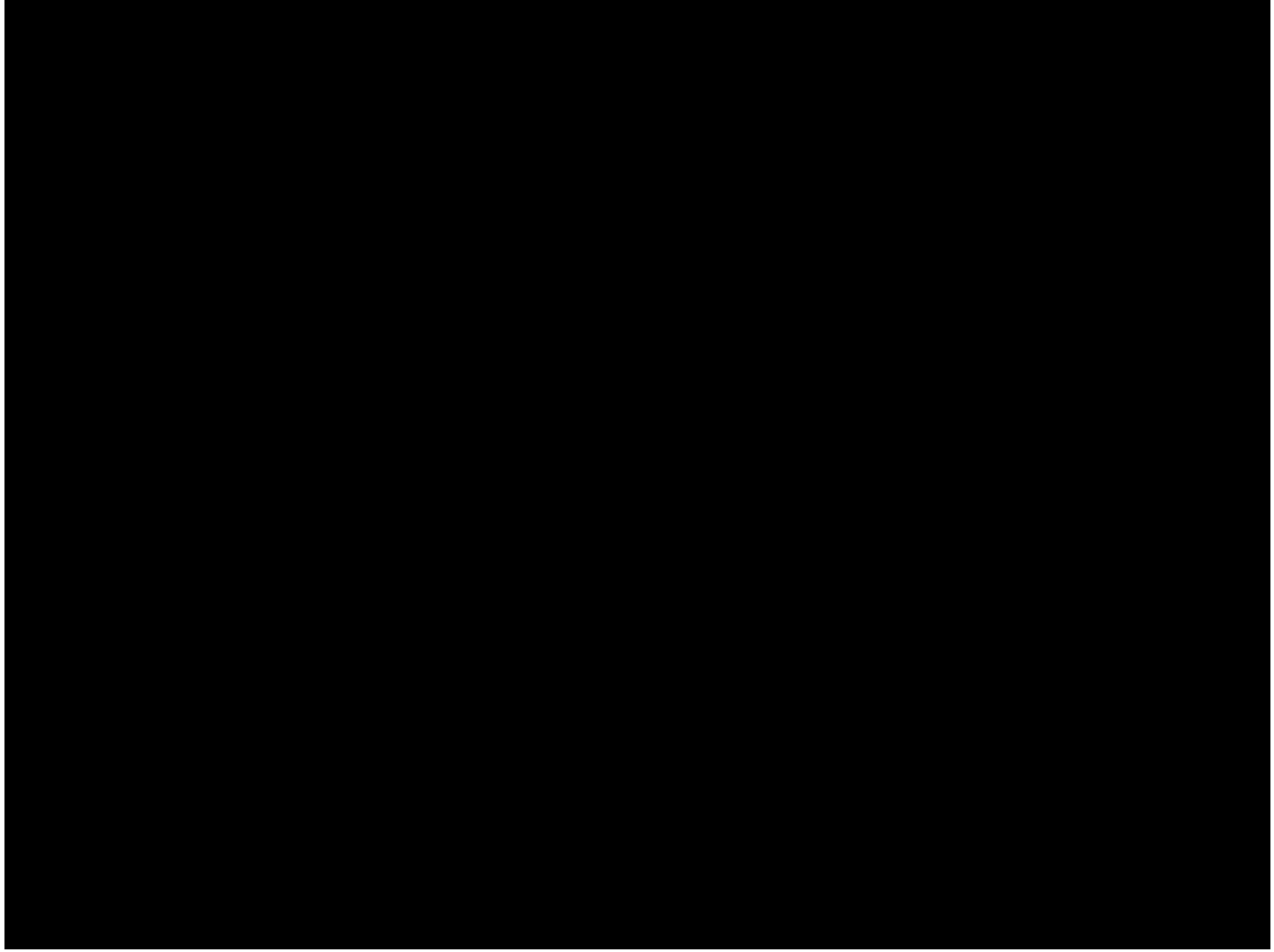
--build <build_directory>
--install <build_directory>

CMake Workflow

cmake -S <source_dir> -B <build_dir>

cmake -S <source_dir> -B <build_dir> -DSOME_OPTION=ON

cmake --build <build_dir>



CMake Configure and Generate

CMake execution occurs over two phases

Configuration: Parsing of the CMakeLists.txt this where commands like option, message, and if are executed.

Generation: This is when generator expression ($<>) are evaluated and all flags, includes, dependencies are computed and written. Happens after the entire configuration process has run

CMake Configure and Generate

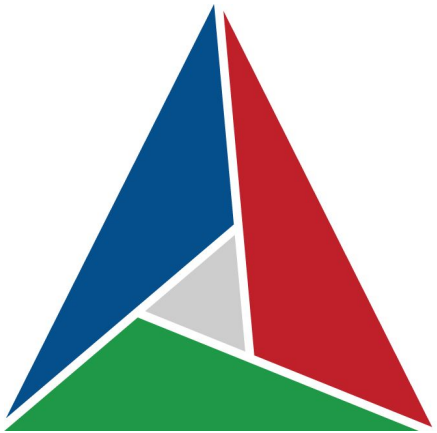
```
cmake_minimum_required(VERSION 3.18 FATAL_ERROR)

project(Demo VERSION 0.1.0 LANGUAGES CXX)

message(STATUS "Hi from configure")

add_executable(demo demo.cpp)
target_link_libraries(demo PRIVATE perf_lib)

string(APPEND CMAKE_CXX_FLAGS "-DOPT_DEFINE")
add_library(perf_lib perf.cpp)
target_compile_options(demo PRIVATE -mcpu=native)
```



CMake

CMake variables and the cache

CMakeCache.txt

Provides a project global variable repository

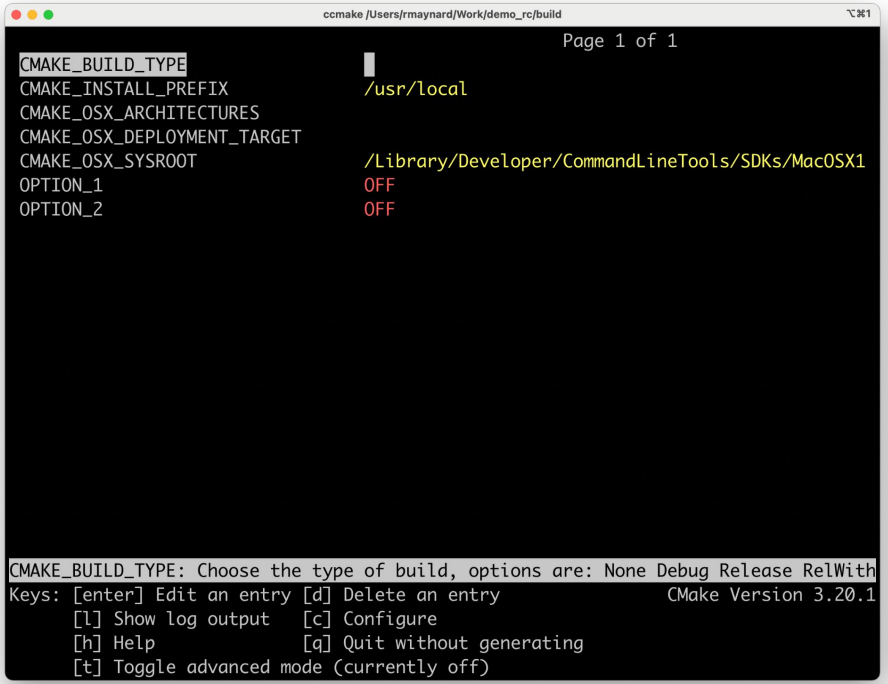
All values are kept from run to run

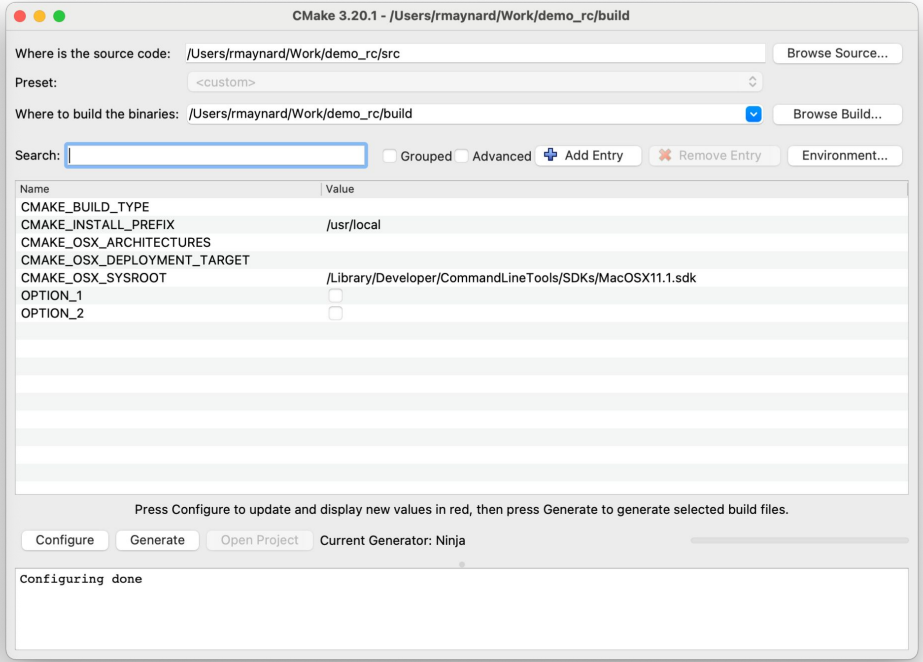
```
cmake -S <src_dir> -B <build_dir> -DOPTION_2=OFF
cmake -S <src_dir> -B <build_dir> -DOPTION_1=OFF
```

cache state is equal to

```
cmake -S <src_dir> -B <build_dir> -DOPTION_1=OFF -DOPTION_2=OFF
```

CMake Variables and the Cache





Variables and the Cache

Dereferences look first for a local variable, then in the cache if there is no local definition for a variable

Local variables hide cache variables

Always prefer local variables to cache variables

Variables and the Cache

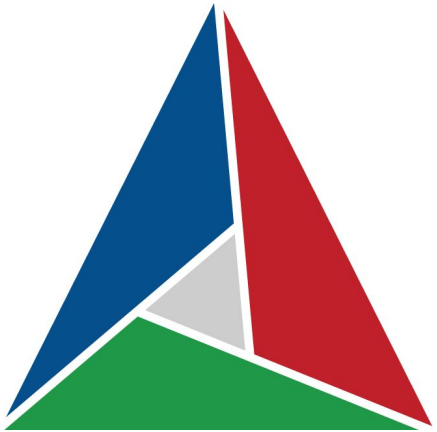
```
option(build_option OFF)
message("build_option value ='${build_option}'")
set(build_option "ON")
message("build_option value ='${build_option}'")
```

```
build_option value ='OFF'
build_option value ='ON'
```

Variables and Scope

Local CMake variables have a scope. Scope is added on add_subdirectory and function

When a new scope is added all existing variables are *captured by value*



CMake

CMake compiler selection

CMake Compiler Selection

CMake caches the compiler used on the first execution of CMake on a per build directory basis

CMake extracts lots of system and compiler information during first configuration, such as include/link directories, language level support (C++14/1720)...

This extraction means that changing the compiler is best done by deleting the CMakeCache.txt and CMakeFiles/

CMake Compiler Selection

Users can specify which specific compiler to use via environment variables or ‘-D’ variables such as CMAKE_CXX_COMPILER

```
CUDACXX=/usr/local/cuda-11.0/bin/nvcc CUDAHOSTCXX=g++-8 cmake -S src -
```

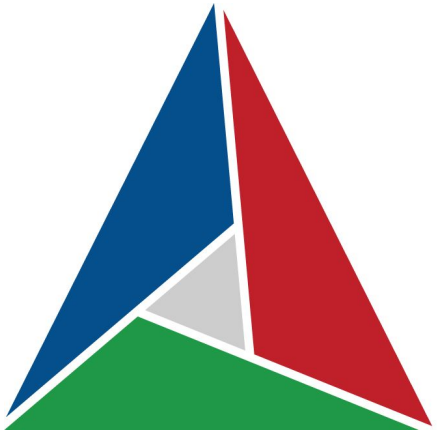
Environment Variables for Languages

- ASM<DIALECT>
- ASM<DIALECT>FLAGS
- CC
- CFLAGS
- CSFLAGS
- CUDAARCHS
- CUDACXX
- CUDAFLAGS
- CUDAHOSTCXX
- CXX
- CXXFLAGS
- FC
- FFLAGS
- HIPCXX
- HIPFLAGS
- ISPC
- ISPCFLAGS
- OBJC
- OBJCXX
- RC
- RCFLAGS
- SWIFTC

CMake Compiler Selection

If no environment variable or compiler variable has been specified CMake searches the PATH for the following programs

1. c++
2. g++
3. aCC
4. cl
5. bcc
6. xLC
7. icpx
8. icx
9. clang++



CMake

CMake changing compiler flags

Adding Compiler Flags

How should we specify project compiler flags?

```
set(CMAKE_CXX_FLAGS "-Wall")
set(CMAKE_CUDA_FLAGS "-Xcompiler=-Wall")
```

```
string(APPEND CMAKE_CXX_FLAGS " -Wall")
string(APPEND CMAKE_CUDA_FLAGS " -Xcompiler=-Wall")
```

```
list(APPEND CMAKE_CXX_FLAGS " -Wall")
list(APPEND CMAKE_CUDA_FLAGS " -Xcompiler=-Wall")
```

Adding Compiler Flags

How should we specify project compiler flags?

```
string(APPEND CMAKE_CXX_FLAGS " -Wall")
string(APPEND CMAKE_CUDA_FLAGS " -Xcompiler=-Wall")
```

CMAKE_<LANG>_FLAGS is required to be a single string, and not a list

We should only add to it, not overwrite

Adding Compiler Flags

Making flags only apply to given compiler

```
if(CMAKE_FORTRAN_COMPILER_ID STREQUAL "NVHPC")
     string(APPEND CMAKE_FORTRAN_FLAGS " -Mvect=simd")
endif()
```

All Compiler Ids:

https://cmake.org/cmake/help/latest/variable/CMAKE_LANG_COMPILER_ID.html

NVHPC => nvc, nvc++, nvfortran

NVIDA => only nvcc

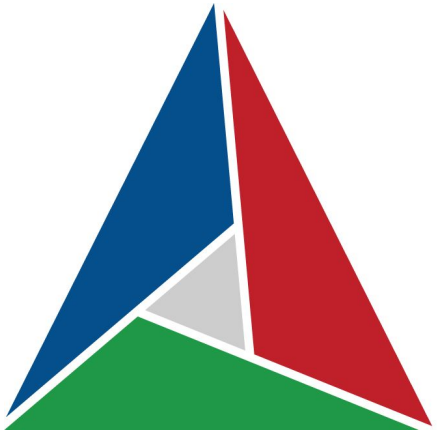
ARMClang

ARMCC

Adding Compiler Flags

More modern CMake project use generator expressions to apply compile flags.

```
set(gcc_like_flags -march=native)
set(nvhpc_flags -Mvect=simd)
add_library(developer_flags INTERFACE)
target_compile_options(developer_flags INTERFACE
  # Flags for CXX GCC builds
    $<$<LANG_AND_COMPILE_ID:CXX,GCC,Clang,ARMClang>:${gcc_like_flags}>
  # Flags for CXX NVHPC builds
    $<$<LANG_AND_COMPILE_ID:CXX,NVHPC>:${nvhpc_flags}>)
```



CMake

Debugging CMake

CMake Command Line Arguments

--system-information = Dump information about this system.

--trace = Put cmake in trace mode.
--trace-expand = Put cmake in trace mode with variable expansion.
--trace-source=<files> = Trace only these CMake files/modules.

--system-information

Great when you need more information from another person.

Effectively combines all information in CMakeCache.txt and CMakeFiles/ in a single file when executed from a projects build directory.

--trace and --trace-expand

Outputs every single line of CMake executed

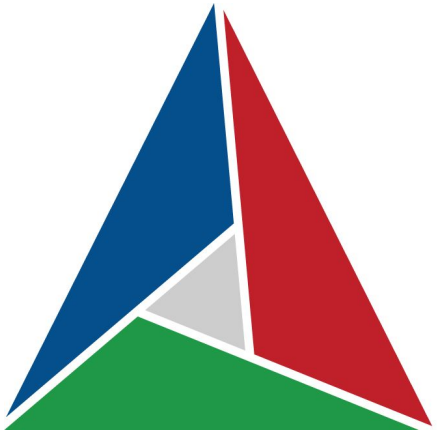
Helps if you need to debug why a file is being included. With trace expand you can see where a compile flag or include is being introduced

variable_watch

CMake command that allows you watch each read or write of a variable

```
function(pretty_print variable access value current_list_file stack)
  if(access STREQUAL MODIFIED_ACCESS)
    message(STATUS "${variable} value updated to [\"${value}\"] from => ${current_list_file}")
  endif()
endfunction()
variable_watch(CMAKE_CXX_FLAGS pretty_print)
```

```
-- CMAKE_CXX_FLAGS value updated to ["-good_flag"] from => /Users/rmaynard/Work/demo_rc/src/CMakeLists.txt
-- CMAKE_CXX_FLAGS value updated to ["-good_flag -bad_flag"] from => /Users/rmaynard/Work/demo_rc/src/CMakeLists.txt
```



CMake

Find Modules

Using Find Modules

One of CMake strengths is the find_package infrastructure

CMake provides 150 find modules

- cmake --help-module-list
- https://cmake.org/cmake/help/latest/manual/cmake-modules.7.html

```
find_package(PythonInterp)
find_package(TBB REQUIRED)
```

Using Find Modules

- Modern approach: packages construct import targets which combine necessary information into a target.
- Classic CMake: when a package has been found it will define the following:
  - <NAME>_FOUND
  - <NAME>_INCLUDE_DIRS
  - <NAME>_LIBRARIES

Using Find Modules

Our library “trunk” needs PNG

```
find_package(PNG REQUIRED)
add_library(trunk SHARED trunk.cxx)
```

Preferred Modern CMake approach:

```
target_link_libraries(trunk PRIVATE PNG::PNG)
```

Historical (Classic) approach:

```
target_link_libraries(trunk ${PNG_LIBRARIES})
include_directories(trunk ${PNG_INCLUDE_DIRS})
```

Using Config Modules

`find_package` also supports config modules

- Config modules are generated by the CMake `export` command
- Will generate import targets with all relevant information, removing the need for consuming projects to write a find module

Understanding Find Modules Searches

CMake's `find_package` uses the following pattern:

- <PackageName>_ROOT from cmake, than env [3.12]
- CMAKE_PREFIX_PATH from cmake
- <PackageName>_DIR from env
- CMAKE_PREFIX_PATH from env
- Any path listed in

```
find_package(PNG HINTS /opt/png/)
```

Understanding Find Modules Searches

- PATH from env
- paths found in the CMake User Package Registry
- System paths as defined in the toolchain/platform
  - CMAKE_SYSTEM_PREFIX_PATH
- Any path listed in

```
find_package(PNG PATHS /opt/png/)
```

Find Module Variables

In general all the search steps can be selectively disabled. For example to disable environment paths:

```
find_package(<package> NO_SYSTEM_ENVIRONMENT_PATH)
```

You can disable all search locations except HINTS and PATHS with:

```
find_package(<package> PATHS paths... NO_DEFAULT_PATH)
```

Direct Find Modules Searches

CMAKE_PREFIX_PATH

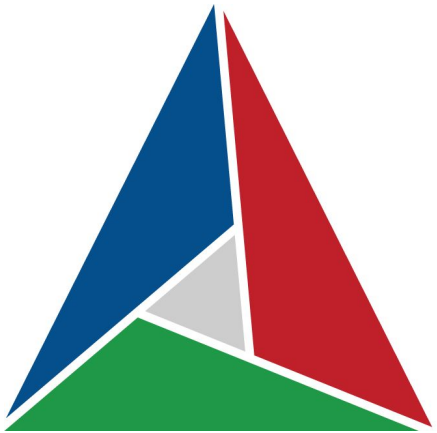
- Prefix used by find_package as the second search path

```
<prefix>/                                                    (W)
<prefix>/(cmake|CMake)/                                      (W)
<prefix>/<name>*/                                            (W)
<prefix>/<name>*/(cmake|CMake)/                              (W)
<prefix>/(lib/<arch>|lib|share)/cmake/<name>*/               (U)
<prefix>/(lib/<arch>|lib|share)/<name>*/                     (U)
<prefix>/(lib/<arch>|lib|share)/<name>*/(cmake|CMake)/       (U)
<prefix>/<name>*/(lib/<arch>|lib|share)/cmake/<name>*/       (W/U)
<prefix>/<name>*/(lib/<arch>|lib|share)/<name>*/             (W/U)
<prefix>/<name>*/(lib/<arch>|lib|share)/<name>*/(cmake|CMake)/  (W/U)
```

Direct Find Modules Searches

<PackageName>_ROOT

- Prefix used by find_package to start searching for the given package
- The package root variables are maintained as a stack so if called from within a find module, root paths from the parent's find module will also be searched after paths for the current package.



CMake

Debugging Find Modules

Debugging Find Modules [3.17+]

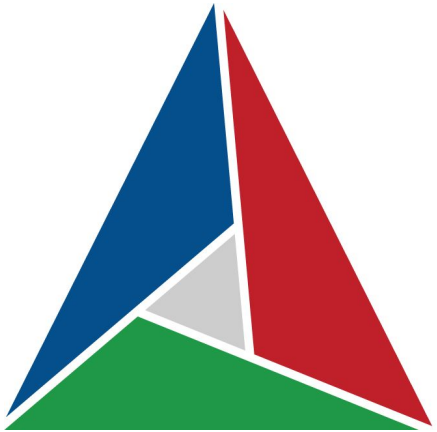
```
find_package(XYZ REQUIRED)
```

```
cmake --find-debug .
…
find_package considered the following paths for XYZ.cmake
  /opt/cmake/.../Modules/FindXYZ.cmake
The file was not found.
find_package considered the following locations for the Config module:
  /home/robert/.local/XYZConfig.cmake
  /home/robert/.local/xyz-config.cmake
  …
  /opt/cmake/XYZConfig.cmake
  /opt/cmake/xyz-config.cmake
The file was not found.
```

Debugging Find Calls [3.17+]

```
set(CMAKE_FIND_DEBUG_MODE 1)
find_library(PNG_LIBRARY_RELEASE
             NAMES png png_static)
```

```
find_library called with the following settings:
     VAR: PNG_LIBRARY_RELEASE
     NAMES: "png" "png_static"
     Documentation: Path to a library.
...
find_library considered the following locations:
     /home/robert/.local/bin/(lib)png(\.so|\.a)
     /usr/local/cuda/bin/(lib)png(\.so|\.a)
     …
The item was not found.
```



CMake

Questions