CMAKE

Sandy Carter - October 2019 - Guest Lecture

HISTORY

- Development began in 1999
- Inspired by `pcmaker` by Ken Martin
- Initial release: 2000
- Old CMake (Version 2.*): April 2005
  - `UPPER_CASE_COMMANDS()`
- Modern CMake (Version 3.*): June 2014
  - `lower_case_commands()`
  - `target_*` commands

HISTORY

- Development began in 1999
- Inspired by `pcmaker` by Ken Martin
- Initial release: 2000
- Old CMake (Version 2.*): April 2005
  - `UPPER_CASE_COMMANDS()`
- Modern CMake (Version 3.*): June 2014
  - `lower_case_commands()`
  - `target_*` commands
  - You should be using Modern CMake principles

WHY USE CMAKE?

OPEN SOURCE

License BSD 3-clause

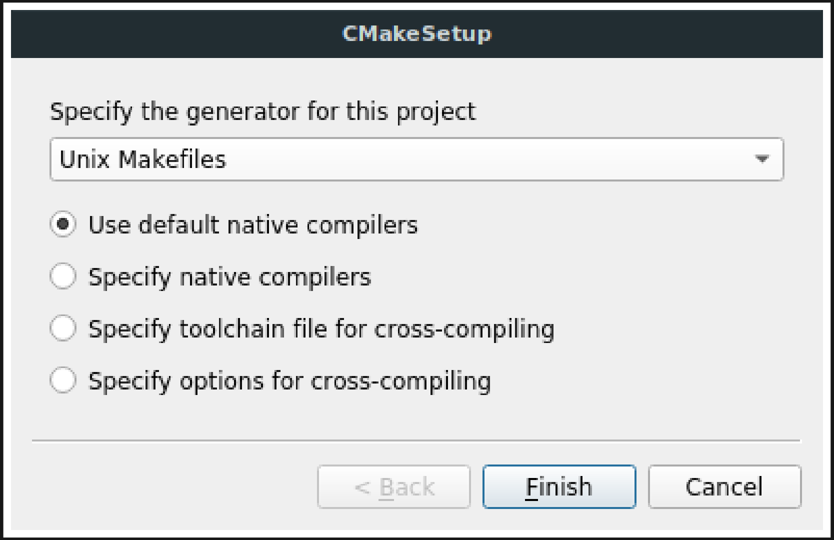
CROSS-PLATFORM

- Windows 32-bit and 64-bit
- MinGW/MSYS2 + Cygwin
- Mac OSX 10.7 or later
- Linux (Virtually all distros and architectures)
- FreeBSD and more!

MULTIPLE GENERATORS

No need to retarget Windows SDK

```
$ cmake -G "Name of generator" ...
```



CMakeSetup

Specify the generator for this project

Unix Makefiles

- [✓ Use default native compilers]
- Specify native compilers
- Specify toolchain file for cross-compiling
- Specify options for cross-compiling

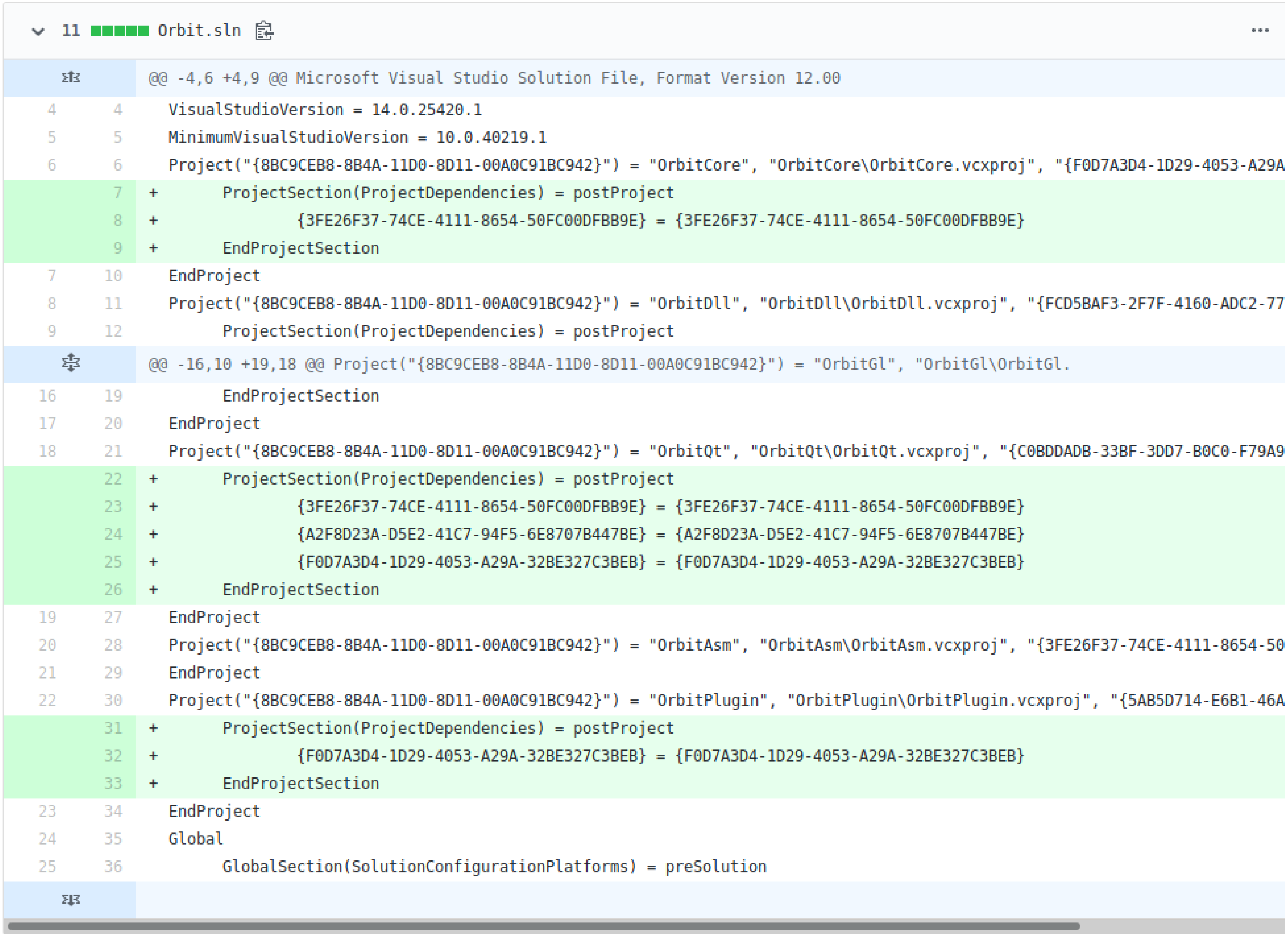
MULTIPLE GENERATORS

- Makefiles (Borland, MSYS/MinGW, NMake, Unix, Watcom)
- Visual Studio .sln (6, 7, 2005, 2010, 2012, 2013, 2015, 2017, 2019)
- Ninja
- Clion (Native)
- CodeBlocks
- Code Lite, Eclipse, Kate, Sublime Text, etc.

CROSS COMPILE WITH TOOLSETS

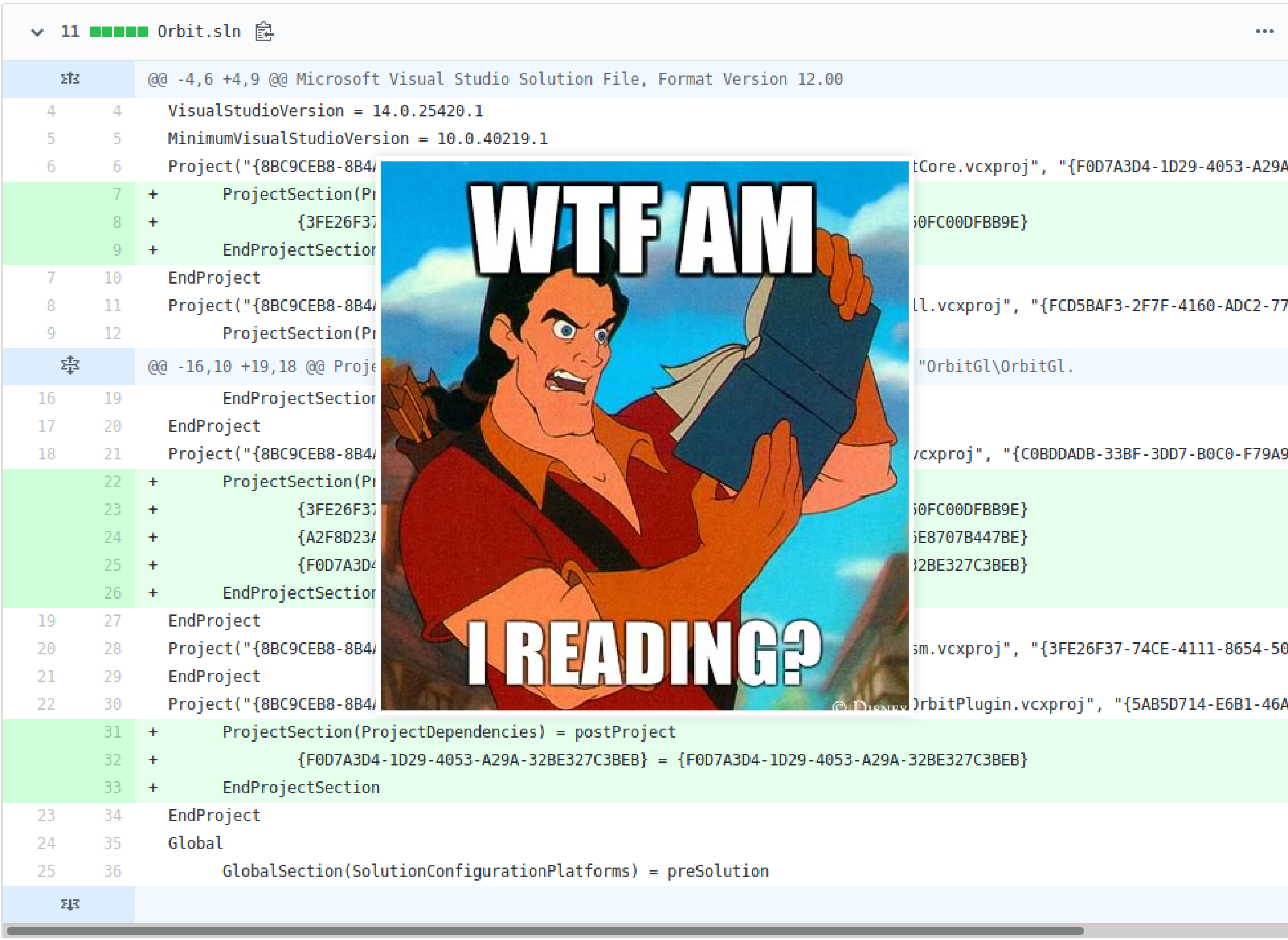
- MinGW-64
- Arduino
- Docker
- IOS
- Android
- WebAssembly
- AVR, ARM, PPC, etc

PROJECT CONFIGURATION AS CODE



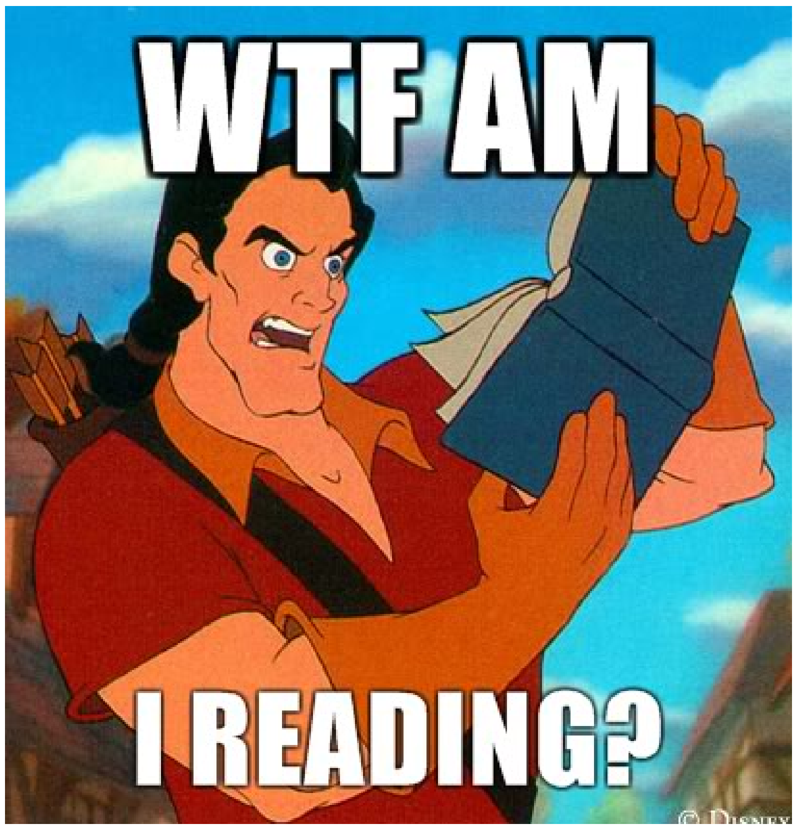
```
11 Orbit.sln

@@ -4,6 +4,9 @@ Microsoft Visual Studio Solution File, Format Version 12.00
  VisualStudioVersion = 14.0.25420.1
  MinimumVisualStudioVersion = 10.0.40219.1
  Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "OrbitCore", "OrbitCore\OrbitCore.vcxproj", "{F0D7A3D4-1D29-4053-A29A
+       ProjectSection(ProjectDependencies) = postProject
+               {3FE26F37-74CE-4111-8654-50FC00DFBB9E} = {3FE26F37-74CE-4111-8654-50FC00DFBB9E}
+       EndProjectSection
  EndProject
  Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "OrbitDll", "OrbitDll\OrbitDll.vcxproj", "{FCD5BAF3-2F7F-4160-ADC2-77
        ProjectSection(ProjectDependencies) = postProject
@@ -16,10 +19,18 @@ Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "OrbitGl", "OrbitGl\OrbitGl.
        EndProjectSection
  EndProject
  Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "OrbitQt", "OrbitQt\OrbitQt.vcxproj", "{C0BDDADB-33BF-3DD7-B0C0-F79A9
+       ProjectSection(ProjectDependencies) = postProject
+               {3FE26F37-74CE-4111-8654-50FC00DFBB9E} = {3FE26F37-74CE-4111-8654-50FC00DFBB9E}
+               {A2F8D23A-D5E2-41C7-94F5-6E8707B447BE} = {A2F8D23A-D5E2-41C7-94F5-6E8707B447BE}
+               {F0D7A3D4-1D29-4053-A29A-32BE327C3BEB} = {F0D7A3D4-1D29-4053-A29A-32BE327C3BEB}
+       EndProjectSection
  EndProject
  Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "OrbitAsm", "OrbitAsm\OrbitAsm.vcxproj", "{3FE26F37-74CE-4111-8654-50
  EndProject
  Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "OrbitPlugin", "OrbitPlugin\OrbitPlugin.vcxproj", "{5AB5D714-E6B1-46A
+       ProjectSection(ProjectDependencies) = postProject
+               {F0D7A3D4-1D29-4053-A29A-32BE327C3BEB} = {F0D7A3D4-1D29-4053-A29A-32BE327C3BEB}
+       EndProjectSection
  EndProject
  Global
        GlobalSection(SolutionConfigurationPlatforms) = preSolution
```

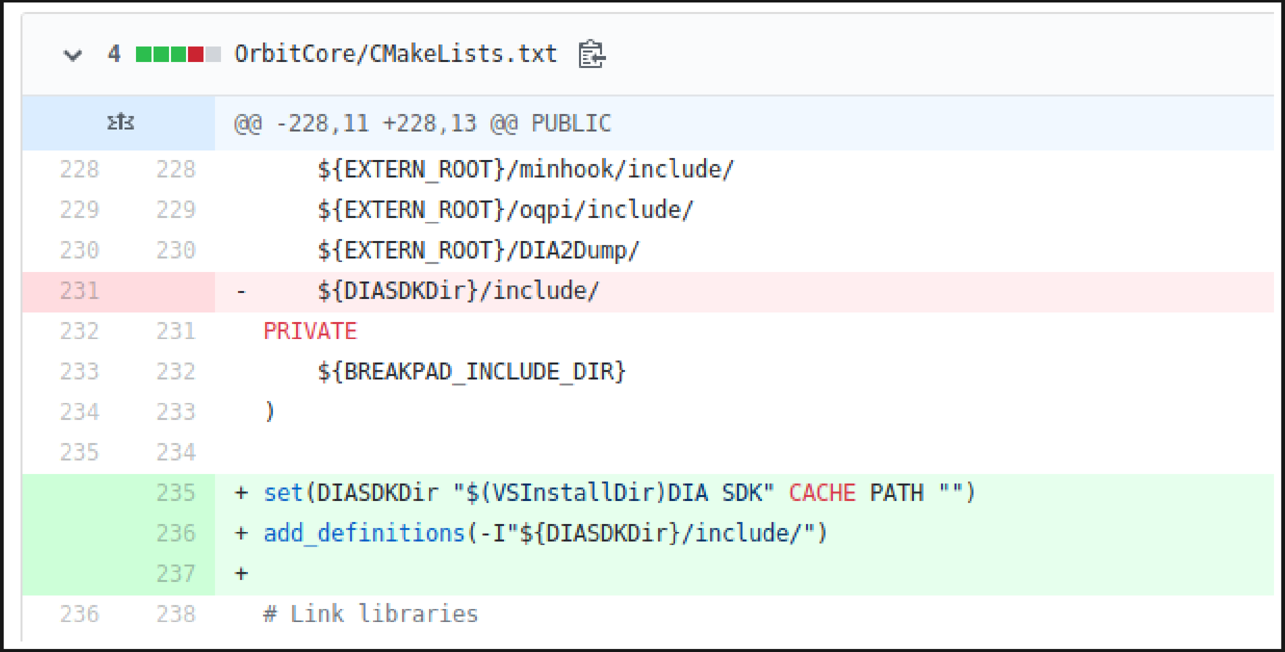


```
11 Orbit.sln

@@ -4,6 +4,9 @@ Microsoft Visual Studio Solution File, Format Version 12.00
  VisualStudioVersion = 14.0.25420.1
  MinimumVisualStudioVersion = 10.0.40219.1
  Project("{8BC9CEB8-8B4A                                      tCore.vcxproj", "{F0D7A3D4-1D29-4053-A29A
+         ProjectSection(Pr
+                 {3FE26F37                                    50FC00DFBB9E}
+         EndProjectSectio
  EndProject
  Project("{8BC9CEB8-8B4A                                      ll.vcxproj", "{FCD5BAF3-2F7F-4160-ADC2-77
          ProjectSection(Pr
@@ -16,10 +19,18 @@ Proje                                            "OrbitGl\OrbitGl.
          EndProjectSectio
  EndProject
  Project("{8BC9CEB8-8B4A                                      vcxproj", "{C0BDDADB-33BF-3DD7-B0C0-F79A9
+         ProjectSection(Pr
+                 {3FE26F37                                    50FC00DFBB9E}
+                 {A2F8D23A                                    5E8707B447BE}
+                 {F0D7A3D4                                    32BE327C3BEB}
+         EndProjectSectio
  EndProject
  Project("{8BC9CEB8-8B4A                                      sm.vcxproj", "{3FE26F37-74CE-4111-8654-50
  EndProject
  Project("{8BC9CEB8-8B4A                                      OrbitPlugin.vcxproj", "{5AB5D714-E6B1-46A
+         ProjectSection(ProjectDependencies) = postProject
+                 {F0D7A3D4-1D29-4053-A29A-32BE327C3BEB} = {F0D7A3D4-1D29-4053-A29A-32BE327C3BEB}
+         EndProjectSection
  EndProject
  Global
          GlobalSection(SolutionConfigurationPlatforms) = preSolution
```



PROJECT CONFIGURATION AS CODE



4 OrbitCore/CMakeLists.txt

```
@@ -228,11 +228,13 @@ PUBLIC
          ${EXTERN_ROOT}/minhook/include/
          ${EXTERN_ROOT}/oqpi/include/
          ${EXTERN_ROOT}/DIA2Dump/
-         ${DIASDKDir}/include/
      PRIVATE
          ${BREAKPAD_INCLUDE_DIR}
      )

+ set(DIASDKDir "$(VSInstallDir)DIA SDK" CACHE PATH "")
+ add_definitions(-I"${DIASDKDir}/include/")
+
  # Link libraries
```

https://github.com/pierricgimmig/orbitprofiler

USING CMAKE

STAGES

1. Configure
    - Multi-platform select, options, packages
    - `CMakeCache.txt`
2. Generate
3. Build
4. Install
5. Test, Package, etc

BUILD TYPES

1. Debug
2. Release
3. RelWithDebInfo
4. MinSizeRel
5. Add your own!

CREATING A CMAKE PROJECT

```
# CMakeLists.txt

# Set the minimum required version of CMake for this project.
# Error will be raised if version is too low.
cmake_minimum_required(VERSION 3.10 FATAL_ERROR)

# Give the project a name.
# This is analogous to a solution name in VS
project(hello_cmake)
```

BASIC CONFIGURATION (EXE)

```cmake
cmake_minimum_required(VERSION 3.10 FATAL_ERROR)

project(hello_cmake)

# Define an executable
add_executable(
  # Executable name
  hello_cmake
  # Files required to compile
  main.cpp other.cpp
)
```

BASIC CONFIGURATION (LIB)

```cmake
cmake_minimum_required(VERSION 3.10 FATAL_ERROR)

project(hello_cmake)

# Define a library
add_library(
  # Library name
  hello_cmake_library
  # Can be SHARED (.dll/.so) or STATIC (.lib/.a)
  SHARED
  # Files required to compile
  hello_cmake_lib.cpp other.cpp
)
```

VARIABLES

```cmake
cmake_minimum_required(VERSION 3.10 FATAL_ERROR)
project(hello_cmake)

file(
  # GLOB to match a regular expression
  GLOB
  # Name of output variable
  LIBRARY_SOURCE_FILES
  # Regular expression pattern
  lib/*.cpp
)

# Variables values are accessed by putting their names in ${}
add_library(hello_cmake_library SHARED ${LIBRARY_SOURCE_FILES}
```

LINKING

```cmake
file(GLOB LIBRARY_SOURCE_FILES lib/*.cpp)
add_library(hello_cmake_library SHARED ${LIBRARY_SOURCE_FILES}

file(GLOB APP_SOURCE_FILES app/*.cpp)
add_executable(hello_cmake ${APP_SOURCE_FILES})

# Declare hello_cmake_library as a dependency and link with it
target_link_libraries(
  # Target in question: the executable
  hello_cmake
  # Inheritance (PRIVATE, PUBLIC or INTERFACE)
  PRIVATE  # targets depending on this will not inherit
    hello_cmake_library
)
```

INHERITANCE

```
add_library(hello_cmake_library SHARED ${LIBRARY_SOURCE_FILES}
# Set different include directories for library and clients
target_include_directories(hello_cmake_library
  PRIVATE  # Applies to this target only
    lib/include/hello_cmake_library
  INTERFACE  # Applies to clients but not target
    lib/include # clients forced to #include <hello_cmake_libr
  PUBLIC  # Applies to both target and clients
    ${OTHER_LIB_INCLUDE_DIRECTORIES}
)

add_executable(hello_cmake ${APP_SOURCE_FILES})
# Linking with library inherits include directories
target_link_libraries(hello_cmake PRIVATE hello_cmake_library)
```

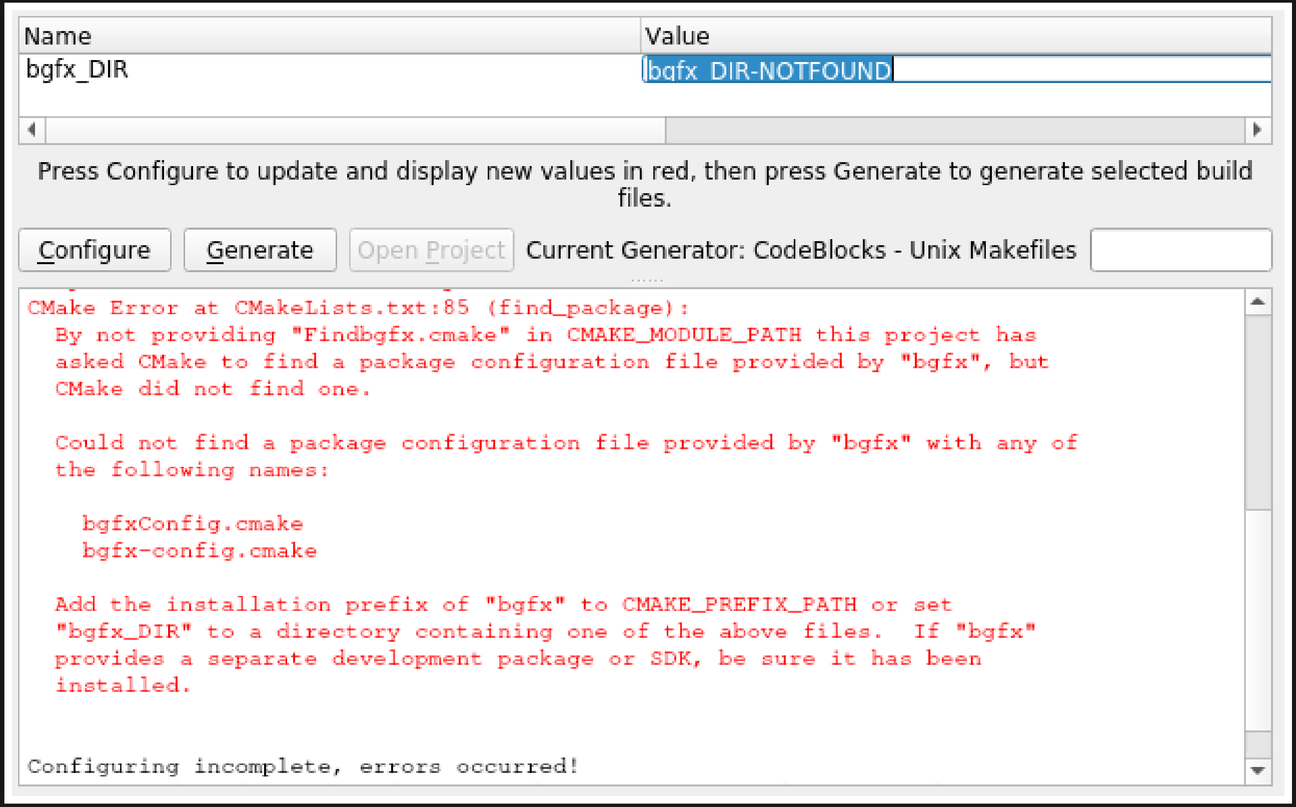
EXTERNAL PACKAGES

```
cmake_minimum_required(VERSION 3.10 FATAL_ERROR)
project(hello_cmake)

# Find OpenGL package installed on the system
# These are imported target aliases which means cmake won't tr
# compile them.
# Looks for cmake/Modules/FindOpenGL.cmake, this file defines
# to find the lib and include directories and creates the alia
find_package(OpenGL REQUIRED COMPONENTS GL GLU)

add_executable(hello_cmake main.cpp other.cpp)
# Link with both components from the OpenGL namespace
target_link_libraries(hello_cmake PRIVATE OpenGL::GL OpenGL::G
```

PACKAGE NOT FOUND



bgfx_DIR   bgfx_DIR-NOTFOUND

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: CodeBlocks - Unix Makefiles

```
CMake Error at CMakeLists.txt:85 (find_package):
  By not providing "Findbgfx.cmake" in CMAKE_MODULE_PATH this project has
  asked CMake to find a package configuration file provided by "bgfx", but
  CMake did not find one.

  Could not find a package configuration file provided by "bgfx" with any of
  the following names:

    bgfxConfig.cmake
    bgfx-config.cmake

  Add the installation prefix of "bgfx" to CMAKE_PREFIX_PATH or set
  "bgfx_DIR" to a directory containing one of the above files.  If "bgfx"
  provides a separate development package or SDK, be sure it has been
  installed.


Configuring incomplete, errors occurred!
```

FIND PACKAGE SEARCH PROCEDURE

Windows

```
C:/Program Files/
C:/Program Files/(cmake|CMake)/
C:/Program Files/<name>*/
C:/Program Files/<name>*/(cmake|CMake)/
```

Unix

```
/usr/(lib/<arch>|lib*|share)/cmake/<name>*/
/usr/(lib/<arch>|lib*|share)/<name>*/
/usr/(lib/<arch>|lib*|share)/<name>*/(cmake|CMake)/
```

OTHER USEFUL COMMANDS

```
# Add precompiler definitions
target_compile_definitions(<target> <INTERFACE|PUBLIC|PRIVATE>

# Add compiler-specific options (e.g. -Wall)
target_compile_options(<target> [BEFORE] <INTERFACE|PUBLIC|PRI

# Add cross-compiler features (e.g. cxx_constexpr)
target_compile_features(<target> <PRIVATE|PUBLIC|INTERFACE> <f

# Find an executable on the system
find_program (<VAR> name1 [path1 path2 ...])

# Generate a target using unusual commands (e.g. compiling sha
add_custom_target(Name [ALL] [command1 [args1...]] ...)
```

USING A PACKAGE MANAGER

- `FetchContent` (CMake >= 3.11): https://cmake.org/cmake/help/latest/module/Fetc
- Vcpkg: https://github.com/Microsoft/vcpkg
- Hunter: https://github.com/cpp-pm/hunter

DIFFERENT VERSIONS OF CMAKE

Platform	Version
Latest Release	3.15.4
Arch Linux (rolling)	3.15.4
OSX (Homebrew)	3.15.4
Visual Studio 2019	3.14.1
Ubuntu disco (19.04)	3.13.4
Visual Studio 2017	3.12.1
Ubuntu bionic (18.04 LTS)	3.10.2

RESOURCES

OFFICIAL RESOURCES

- Website: https://cmake.org
- Source: https://gitlab.kitware.com/cmake
- Documentation: https://cmake.org/cmake/help
- Wiki: https://gitlab.kitware.com/cmake/community/wikis
- Mailing list: https://cmake.org/mailing-lists

RESOURCES

GUIDES

- Modern CMake: https://cliutils.gitlab.io/modern-cm
- Effective Modern CMake: https://gist.github.com/mbinna/c61dbb39bca0e4fb
- CMake Anti-Antipatterns: https://blog.kevinwmatth anti-antipatterns
- Awesome CMake https://github.com/onqtam/awes

QUESTIONS?