﻿CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice. CMake is quite sophisticated: it is possible to support complex environments requiring system configuration, pre-processor generation, code generation, and template instantiation.

[CMake Wiki](http://www.cmake.org/Wiki/CMake)

CMake C language solution

Let's have a look at a simple example of C language CMake solution.

|  |
| --- |
| FooBar/  | |-- CMakeLists.txt //the main CMakeLists.txt. CMake must start here  | |-- foo/  | | |-- CMakeLists.txt  | | |-- config.h.in //for code generation demonstration  | | |-- Globals.h //export macros for WIN & MAC  | | |-- foo.h  | `-|-- foo.c   `|- bar/    | |-- CMakeLists.txt    `-|- bar.c |

**FooBar dir:**

Foobar/CMakeLists.txt:

|  |
| --- |
| cmake\_minimum\_required(VERSION 2.8)  #declaring C language project named FooBar  project(FooBar C)      #these variables will be used for generation of config.h file  set(FOOBAR\_MAJOR\_VERSION 0)  set(FOOBAR\_MINOR\_VERSION 1)  set(FOOBAR\_PATCH\_VERSION 0)  set(FOOBAR\_VERSION    ${FOOBAR\_MAJOR\_VERSION}.${FOOBAR\_MINOR\_VERSION}.${FOOBAR\_PATCH\_VERSION})    #sets the solution binaries output directory  set(CMAKE\_RUNTIME\_OUTPUT\_DIRECTORY "${PROJECT\_BINARY\_DIR}/bin")    # set up include-directories  include\_directories(    "${PROJECT\_SOURCE\_DIR}"   # to find foo/foo.h    "${PROJECT\_BINARY\_DIR}")  # to find foo/config.h - will be generated from FooBar\foo\config.h.in      #user can add another sub directories here  set(MY\_SUB\_DIRS   foo   bar  )    # Add sub-directories  foreach(i ${MY\_SUB\_DIRS})  add\_subdirectory(${i})  endforeach() |

**FooBar\foo dir (library):**

FooBar\foo\config.h.in:

|  |
| --- |
| #define FOOBAR\_MAJOR\_VERSION (@FOOBAR\_MAJOR\_VERSION@) //variables are set in main FooBar\CMakeLists.txt  #define FOOBAR\_MINOR\_VERSION (@FOOBAR\_MINOR\_VERSION@)  #define FOOBAR\_PATCH\_VERSION (@FOOBAR\_PATCH\_VERSION@)  #define FOOBAR\_VERSION\_STRING "@FOOBAR\_VERSION@" |

Platform dependent export macros FooBar\foo\Globals.h:

|  |
| --- |
| // Operating System Stuff...  #if defined(\_WINDOWS) || defined(\_WIN32) || defined(\_WIN64)  # define OS\_WIN  #elif defined(\_\_APPLE\_\_) && defined(\_\_MACH\_\_)  # define OS\_MAC  #else  # error "Operating System Not Supported. Only Windows and Mac OS X Supported"  #endif    // Exports...  #if defined(OS\_WIN)  # define FB\_EXPORT \_\_declspec(dllexport)  # define FB\_IMPORT \_\_declspec(dllimport)  # define FB\_EXPORT\_METHOD \_\_declspec(dllexport)  #elif defined(OS\_MAC)  # define FB\_EXPORT \_\_attribute\_\_ ((visibility("default")))  # define FB\_IMPORT  # define FB\_EXPORT\_METHOD extern "C" \_\_attribute\_\_ ((visibility("default")))  #endif |

FooBar\foo\foo.h:

|  |
| --- |
| #ifndef FOOBAR\_FOO\_H  #define FOOBAR\_FOO\_H    #include <foo/config.h>  #include "Globals.h"    FB\_EXPORT void foo(void);    #endif |

FooBar\foo\foo.c:

|  |
| --- |
| #include "foo.h"    #include <stdio.h>    void foo(void)  {   printf("This is foo version %s\n", FOOBAR\_VERSION\_STRING);  } |

FooBar\foo\CMakeLists.txt:

|  |
| --- |
| # Copy a file to another location and modify its contents.  # configure\_file(<input> <output>                   # [COPYONLY] [ESCAPE\_QUOTES] [@ONLY])  configure\_file(config.h.in "${CMAKE\_CURRENT\_BINARY\_DIR}/config.h" @ONLY)  if(MSVC)      #build foo as dynamic library      add\_library(foo SHARED foo.c foo.h Globals.h config.h.in)  elseif(APPLE)      #build foo as MAC OS framework      add\_library(foo FRAMEWORK foo.c foo.h Globals.h config.h.in)  endif()    #set foo's include dirs  set\_target\_properties(foo PROPERTIES    PUBLIC\_HEADER "foo.h;${CMAKE\_CURRENT\_BINARY\_DIR}/config.h") |

**FooBar\bar dir (executable):**

FooBar\bar\bar.c:

|  |
| --- |
| #include "foo/foo.h"    int main(int argc, char\* argv[])  {   foo();   return 0;  } |

FooBar\bar\CMakeLists.txt:

|  |
| --- |
| #Add an executable to the project using the specified source files.  # add\_executable(<name> [WIN32] [MACOSX\_BUNDLE]                  # [EXCLUDE\_FROM\_ALL]                  # source1 source2 ... sourceN)    add\_executable(bar bar.c)    #Link a target to given libraries.  # target\_link\_libraries(<target> [item1 [item2 [...]]]                       # [[debug|optimized|general] <item>] ...)    target\_link\_libraries(bar foo) |

Excluding files

|  |
| --- |
| If you want to exclude certain source files from certain targets you can use  a list and then list(REMOVE\_ITEM...)    set(FOO\_SRCS foo.cc bar.cc)  list(REMOVE\_ITEM FOO\_SRCS bar.cc)  add\_library(foo ${FOO\_SRCS})    Alternatively, only add the unusual source files to the targets you want.    set(FOO\_SRCS foo.cc bar.cc)  if(WHATEVER)     list(FOO\_SRCS APPEND whatever.cc)  endif() |

Generation of native solution files

**Gui version**

1. Run cmake-gui
2. Select root of the solution (eg c:\FooBar)
3. Select location of generated native solution files (eg c:\FooBar\sln)
4. Click "Generate"
5. Select desired solution generator (eg "Visual Studio 10 Win64")
6. Click "Finish"

**Command line version**

cmake.exe -G"Visual Studio 10 Win64" -H<source\_dir> -B<build\_dir>

Building with native compilers

Visual Studio

|  |
| --- |
| Devenv SolutionName /build SolnConfigName [/project ProjName [/projectconfig ProjConfigName]]  eg: devenv "C:\Documents and Settings\someuser\My Documents\Visual Studio\Projects\MySolution\MySolution.sln" /build Debug /project "CSharpWinApp\CSharpWinApp.csproj" /projectconfig Debug |

XCode

Build the application into an .app file:

|  |
| --- |
| xcodebuild -target "${PROJECT\_NAME}" -sdk "${TARGET\_SDK}" -configuration Release |

Package it into an .ipa file:

|  |
| --- |
| /usr/bin/xcrun -sdk iphoneos PackageApplication -v "${RELEASE\_BUILDDIR}/${APPLICATION\_NAME}.app" -o "${BUILD\_HISTORY\_DIR}/${APPLICATION\_NAME}.ipa" --sign "${DEVELOPER\_NAME}" --embed "${PROVISONING\_PROFILE}” |