Cmake Xbox Console App

*This sample is compatible with the Microsoft Game Development Kit (June 2020)*

# Description

This is an example of using the [CMake](https://cmake.org/) cross-platform build system to produce a “Win32 console” application that can be executed on the Xbox hardware using the Microsoft GDK. This is suitable for non-graphics developer unit tests using ‘printf’ style output.



*If you are looking for details on using CMake for building standard Microsoft GDK applications, see* ***CMakeExample*** *and* ***CMakeGDKExample****.*

# Building the sample

Using Visual Studio 2019 select “Open a local folder…” from the New Project Dialog or the “File -> Open -> Folder…” menu command and open the sample folder:

* This requires that you have the “C++ CMake tools for Windows” component installed.

If needed, edit the **XdkEditionTarget** variable (either in the CMakeSettings.json or CMakeList.txt) to ensure you have the correct GDK edition referenced.

The CMake tool should generate the cache automatically upon opening. Otherwise select the CMakeList.txt and select “Generate Cache” from the right-button menu. Then use the “Build -> Rebuild All” menu command. The build products are in the “**bin**” subfolder.

See [Microsoft Docs](https://docs.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio) for more information on CMake in Visual Studio.

*This sample makes use of target\_link\_directories so it requires CMake 3.13 or later. Visual Studio 2017 (15.9 update) includes version 3.12, which is why the instructions are for Visual Studio 2019. You can of course use the CMake tool directly instead of relying on Visual Studio integration. If using Visual Studio 2017, you’ll need to modify the logic in the CMakeList.txt for finding the VC Runtime DLLs.*

The default setup includes the **x64-Debug** and **x64-Release** configurations using Visual C++. You can add the **x64-Clang-Debug** and/or **x64-Clang-Release** configurations to use clang/LLVM instead.

* This requires that you have the “C++ Clang Compiler for Windows” component installed.

*If you press F5, it is attempting to run on the development PC and not the remote console so it may or may not fail. You need to deploy the program per the instructions below to run successfully.*

# Using the sample

To deploy the sample, open an *Xbox Gaming Command Prompt* instance and change to the sample directory:

cd CMakeXboxConsoleApp\out\build\<config>

xbcp bin\Console\\*.exe xd:\

xbcp bin\Console\\*.dll xd:\

To run the sample:

xbrun /O D:\CMakeXboxConsoleApp.exe

The program will run in the context of the System OS.

If you wish to run in the context of the Game OS instead, you can use a similar procedure. First start by running a Game OS title on the target console. A good candidate is to use Visual Studio’s New Project dialog and create a default “Direct3D 12 Xbox Game project” with the Microsoft GDK. Build and deploy it, and leave it running.

Then use:

xbcp /x/title bin\Console\\*.exe xd:\

xbcp /x/title bin\Console\\*.dll xd:\

To run the sample:

xbrun /x/title /O D:\CMakeXboxConsoleApp.exe

Keep in mind this works by injecting a process into the Game OS VM. Multi-process game titles are not supported at this time, and several components including graphics, audio, and GameRuntime are not tested nor support for use in multiple process scenarios. It is also a good idea to keep the ‘hosting’ title very simple and limit its use of CPU resources.

# Implementation Details

For PC Desktop, the **CMakeLists.txt** for a Win32 console exe (i.e. /SUBSYSTEM:CONSOLE) would be something like the following:

cmake\_minimum\_required (VERSION 3.13)

project(CMakeExampleWindowsConsole LANGUAGES CXX)

option(BUILD\_USING\_LTCG "Enable Whole Program Optimization" ON)

set(CMAKE\_CXX\_STANDARD 14)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

set(CMAKE\_CXX\_EXTENSIONS OFF)

add\_executable(${PROJECT\_NAME} Main.cpp)

target\_compile\_definitions(${PROJECT\_NAME} PRIVATE "$<$<CONFIG:DEBUG>:\_DEBUG>" "$<$<CONFIG:RELEASE>:NDEBUG>")

target\_compile\_definitions(${PROJECT\_NAME} PRIVATE \_CONSOLE \_UNICODE UNICODE)

# Use Warning Level 4

string(REPLACE "/W3 " "/W4 " CMAKE\_CXX\_FLAGS ${CMAKE\_CXX\_FLAGS})

string(REPLACE "/W3 " "/W4 " CMAKE\_CXX\_FLAGS\_DEBUG ${CMAKE\_CXX\_FLAGS\_DEBUG})

string(REPLACE "/W3 " "/W4 " CMAKE\_CXX\_FLAGS\_RELEASE ${CMAKE\_CXX\_FLAGS\_RELEASE})

# If not using typeid or dynamic\_cast, we can disable RTTI to save binary size

string(REPLACE "/GR " "/GR- " CMAKE\_CXX\_FLAGS ${CMAKE\_CXX\_FLAGS})

string(REPLACE "/GR " "/GR- " CMAKE\_CXX\_FLAGS\_DEBUG ${CMAKE\_CXX\_FLAGS\_DEBUG})

string(REPLACE "/GR " "/GR- " CMAKE\_CXX\_FLAGS\_RELEASE ${CMAKE\_CXX\_FLAGS\_RELEASE})

target\_compile\_options(${PROJECT\_NAME} PRIVATE /fp:fast /GS /Gy)

if(CMAKE\_CXX\_COMPILER\_ID MATCHES "MSVC")

target\_compile\_options(${PROJECT\_NAME} PRIVATE /permissive- /Zc:\_\_cplusplus)

if(CMAKE\_BUILD\_TYPE MATCHES "Debug")

elseif(BUILD\_USING\_LTCG MATCHES ON)

target\_compile\_options(${PROJECT\_NAME} PRIVATE /GL /Gw)

target\_link\_options(${PROJECT\_NAME} PRIVATE /IGNORE:4075 /LTCG)

endif()

endif()

For the System and Game OS on the Xbox hardware, we must use a different set of link libraries and ensure we don’t pick up any unsupported libraries. We should also enable the proper API partitioning to avoid using unsupported APIs, and this sample ensure you are building with the platform headers and libraries.

The application running on the Xbox hardware also needs to provide the Visual C++ Runtime DLLs it requires, as well as the ucrtbased.lib if it’s built for Debug.

The Xbox “console” CMake in this sample is set up to build the EXE to run on either Project Scarlett or Xbox One hardware. Since we can’t use Direct3D for a console app, we avoid the major API differences in the platforms and can reasonably expect the same EXE to run on both platforms. This is also by the specific XboxOne and Scarlett include/lib paths are not set in the CMakeLists.txt.

If desired, you can enable additional compiler CPU targeting specifically for the Project Scarlett hardware. This is done by setting the build option OPTIMIZE\_FOR\_SCARLETT to ON. The resulting EXE will run as before on Project Scarlett, but will fail to run on Xbox One. To demonstrate this, the sample makes use of the DirectXMath XMVerifyCPUSupport function which performs he relevant CPUID checks.

# Additional Information

For more details on all the complier & linker switches used in this example, see **CMakeExample**.

The CMake project in this sample support an opt-in build option to use Build With/Out Installing (BWOI). If enabled, it requires an ExtractedFolder environment variable which points to the extracted Microsoft GDK created by the *BWOIExample*’s extractgdk.cmd script. It can optionally also have an extracted Windows 10 SDK (19041) for the May 2020 GDK or later. The CMake project do not need the results of the vctargets.cmd script because it doesn’t use the Gaming.\*.x64 MSBuild platforms.

See the **BWOIExample** for more details.

# Known Issues

If you use the clang/LLVM toolset, be sure you are using the Windows 10 SDK (19041) which includes DirectXMath 3.14. In DirectXMath 3.13 and before, the XMVerifyCPUSupport implementation doesn’t build correctly for that toolset. See <https://walbourn.github.io/directxmath-3.14/> for details.

# Version History

|  |  |
| --- | --- |
| **May 2020** | Initial version. |
| **June 2020** | Updated for the June 2020 GDK FAL release. |
| **November 2020** | Cleaned up CMake files, added \_CONSOLE define. |
| **February 2021** | Minor CMake cleanup. |