CMake Example

*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 build an executable with the Microsoft Game Development Kit via the Ninja generator.



*The primary purpose of this sample is to clearly document all the paths and settings required to build for the Gaming.\*.x64 platforms. This replicates much of the functionality that is implemented in the MSBuild rules installed by the GDK. For an alternative method of utilizing CMake via the Visual Studio generator, see* ***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 Desktop, XboxOne, or Scarlett 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 XboxOne and Scarlett 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 for the Xbox One or Scarlett projects, it is attempting to run on the development PC and not the remote console so it will 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 CMakeExample\XboxOne\out\build\<config>\bin

### Push deploy

To do push deploy the ‘loose’ layout:

xbapp deploy Gaming.Xbox.XboxOne.x64

### Run-from-PC

To run the ‘loose’ layout from the PC:

xbapp launch Gaming.Xbox.XboxOne.x64\CMakeExampleXboxOne.exe

### Packaged deployment

To create a package:

makepkg genmap /f chunks.xml /d Gaming.Xbox.XboxOne.x64

makepkg pack /f chunks.xml /lt /d Gaming.Xbox.XboxOne.x64 /pd .

For Desktop packaging, also add /pcto the second command line.

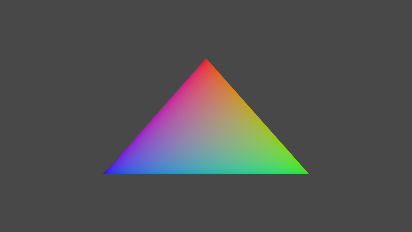
Then install the resulting package to your console (the exact .xvc filename will vary):

xbapp install CMakeExampleXboxOne\_1.0.0.0\_neutral\_\_zjr0dfhgjwvde.xvc

For Desktop, the extension is “.msixvc” (the exact filename will vary):

xbapp install CMakeExampleXboxOne\_1.0.0.0\_neutral\_\_zjr0dfhgjwvde.msixvc

The sample when run creates a device and swapchain, and draws a colored triangle. It has no controls or other behavior.



*For packaging the other versions, see comments at the end of each CMakeLIst.txt for the specific command-line options to use.*

# Implementation Details

For more information on various Visual C++ switches, see the links below:

|  |  |
| --- | --- |
| /GR | <https://docs.microsoft.com/en-us/cpp/build/reference/gr-enable-run-time-type-information> |
| /GS  /RTC  /sdl    /DYNAMICBASE  /NXCOMPAT | <https://aka.ms/msvcsecurity> |
| /DEBUG:fastlink | <https://devblogs.microsoft.com/cppblog/faster-c-build-cycle-in-vs-15-with-debugfastlink/> |
| /EHsc | <https://devblogs.microsoft.com/cppblog/making-cpp-exception-handling-smaller-x64/> |
| /fp | <https://docs.microsoft.com/en-us/cpp/build/reference/fp-specify-floating-point-behavior>  <https://devblogs.microsoft.com/cppblog/game-performance-improvements-in-visual-studio-2019-version-16-2/> |
| /FS | <https://docs.microsoft.com/en-us/cpp/build/reference/fs-force-synchronous-pdb-writes> |
| /GL  /Gw  /LTCG | <https://devblogs.microsoft.com/cppblog/tag/link-time-code-generation/>  <https://devblogs.microsoft.com/cppblog/introducing-gw-compiler-switch/> |
| /Gy | <https://docs.microsoft.com/en-us/cpp/build/reference/gy-enable-function-level-linking> |
| /JMC | <https://devblogs.microsoft.com/cppblog/announcing-jmc-stepping-in-visual-studio/> |
| /permissive- | <https://devblogs.microsoft.com/cppblog/permissive-switch/> |
| /std:c++14 | <https://devblogs.microsoft.com/cppblog/standards-version-switches-in-the-compiler/> |

|  |  |
| --- | --- |
| /Yc  /Yu  /Fp  /FI | <https://docs.microsoft.com/en-us/cpp/build/creating-precompiled-header-files>  <https://devblogs.microsoft.com/cppblog/shared-pch-usage-sample-in-visual-studio/> |
| /Zc:\_\_cplusplus | <https://devblogs.microsoft.com/cppblog/msvc-now-correctly-reports-__cplusplus/> |
| /Zc:preprocessor | <https://devblogs.microsoft.com/cppblog/announcing-full-support-for-a-c-c-conformant-preprocessor-in-msvc/> |
| /Z7, /Zi, /ZI | <https://docs.microsoft.com/en-us/cpp/build/reference/z7-zi-zi-debug-information-format> |

Note that [/Gm](https://docs.microsoft.com/en-us/cpp/build/reference/gm-enable-minimal-rebuild) (Minimal Rebuild) is deprecated and should be removed from projects that still use it.

## Side-by-side toolsets

Per the [Visual C++ blog](https://devblogs.microsoft.com/cppblog/side-by-side-minor-version-msvc-toolsets-in-visual-studio-2019/), you can use older versions of the compiler toolset with the newer version of the Visual Studio IDE. For CMake, you do this via **CMakeSettings.json**. For example, if you want to use the VS 2019 (16.0) version of the compiler, add:

"environments":

[

{

"ClearDevCommandPromptEnvVars": "false"

"VCToolsVersion": "14.20.27508",

}

],

# Additional Information

The CMake projects 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 projects do not need the results of the vctargets.cmd script because they don’t use the Gaming.\*.x64 MSBuild platforms.

See the **BWOIExample** for more details.

# Version History

|  |  |
| --- | --- |
| **November 2019** | Initial version. |
| **February 2020** | Added use of HLSL shaders to the example.  Updated to optionally support BWOI. |
| **April 2020** | Updated with pch support when using CMake 3.16 or later. |
| **May 2020** | Updated to support the May 2020 GDK. |
| **June 2020** | Updated for the June 2020 GDK FAL release. |
| **August 2020** | Updated with side-by-side details. |
| **November 2020** | Added xmem.lib and xg\_\*.lib for Xbox targets.  Cleaned up CMake files. |
| **February 2021** | Minor updates for CMake comments. |
| **April 2021** | Add appnotify.lib to resolve link issues with Desktop target.  Add LargeLogo.png. |
| **June 2021** | General code cleanup. |