CMake GDK 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 Visual Studio generator.



*This sample demonstrates how to use CMake to generate Gaming.\*.x64 platform VC++ project files that use the Microsoft GDK to build.* *For an alternative method of utilizing CMake, see* ***CMakeExample****.*

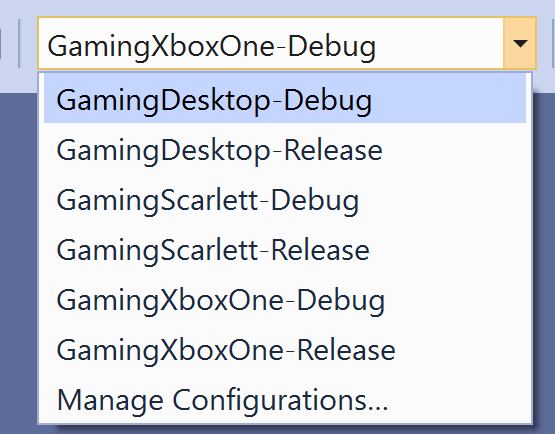
# Building the sample (Visual Studio)

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 (Microsoft.VisualStudio.Component.VC.CMake.Project) installed.

If needed, edit the **XdkEditionTarget** variable (either in the CMakeSettings.json or gxdk\_toolchain.cmake / gxdk\_xs\_toolchain.cmake) 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. Choose which platform to build for in the combo box:



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

# Building the sample (command-line)

You can also generate and build from the command line using a *VS 2019 Developer Command Prompt*:

cd CMakeGDKExample

cmake . -B out -DXdkEditionTarget=210600 -DCMAKE\_TOOLCHAIN\_FILE=.\gxdk\_toolchain.cmake

cmake --build out

# Using the sample (Visual Studio)

Open the generated SLN/VCXPROJ from another instance of Visual Studio:

CMakeGDKExample\out\build\GamingXboxOne-Debug\CMakeGDKExample.sln

If you are using CMake 3.17 or earlier, first use the Configuration Manager to check the “Deploy” checkbox for the *CMakeGDKExample* project.

Then use F5 to deploy/run.

*If you press F5 from the original CMakeLists.txt context, then it will fail to start because the loose layout is not placed inside the ‘bin’ directory. For Gaming.Xbox.\*.x64 configurations, it also is attempting to run on the development PC and not the remote console.*

# Using the sample (command-line)

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

cd CMakeGDKExample\out\build\GamingXboxOne-Debug\Gaming.Xbox.XboxOne.x64

For Desktop, the loose layout is in bin\Gaming.Desktop.x64\Debug

### Push deploy

To do push deploy the ‘loose’ layout:

xbapp deploy Layout\Image\Loose

### Run-from-PC

To run the ‘loose’ layout from the PC:

xbapp launch Layout\Image\Loose\CMakeGDKExample.exe

### Packaged deployment

Edit the Layout\Image\Loose\Microsoft.Config to add a TargetDeviceFamily element (“PC”, “Scarlett”, or ”XboxOne”):

<ExecutableList>

<Executable Name="CMakeGDKExample.exe"

**TargetDeviceFamily="XboxOne"**

Id="Game" />

</ExecutableList>

To create a package:

makepkg genmap /f chunks.xml /d Layout\Image\Loose

makepkg pack /f chunks.xml /lt /d Layout\Image\Loose /pd .

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

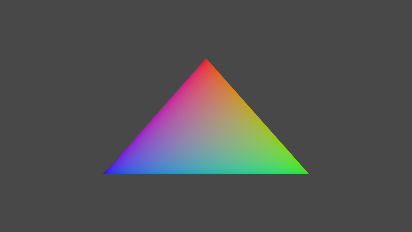
xbapp install CMakeGDKExample\_1.0.0.0\_neutral\_\_8wekyb3d8bbwe\_x.xvc

For Desktop packaging:

makepkg genmap /f chunks.xml /d bin\Gaming.Desktop.x64\Debug

makepkg pack /pc /f chunks.xml /lt /d bin\Gaming.Desktop.x64\Debug /pd .

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



*For packaging the Xbox Series X|S and/or Xbox Release versions, change to the correct directory for the platform & configuration.*

# Implementation Details

The **CMakeExample** uses the “Ninja” generator so does not make use of the Microsoft GDK MSBuild rules. This version uses the “Visual Studio 16 2019 Win64” generator instead which makes use of the Microsoft GDK MSBuild instructions.

The CMake generation makes use of a toolchain file passed as a command-line:

|  |  |
| --- | --- |
| Gaming.Desktop.x64 | -DCMAKE\_TOOLCHAIN\_FILE="grdk\_toolchain.cmake" |
| Gaming.Xbox.XboxOne.x64 | -DCMAKE\_TOOLCHAIN\_FILE="gxdk\_toolchain.cmake" |
| Gaming.Xbox.Scarlett.x64 | -DCMAKE\_TOOLCHAIN\_FILE="gxdk\_xs\_toolchain.cmake" |

All three of these also make use of a custom MSBuild property file gdk\_build.props.

The use of the Gaming.\*.x64 MSBuild rules with the GDK handles the MicrosoftGame.Config localization, placing the CRT files into the layout, etc.

CMake can’t leverage the FXCCompile MSBuild target for shaders, so CMakeLists.txt runs DXC as a custom target. The toolchain is responsible for finding the correct version of the shader compile, which is why the gxdk\_toolchain.cmake and gxdk\_xs\_toolchain.cmake need the **XdkTargetEdition** variable.

In order to support *Build Without Install* (BWOI) with the resulting CMake, you need (a) to explicitly set **GDK\_DXCTool** to point to the proper DXC.EXE for the platform you are building, and (b) to make use of the Directory.Build.props solution detailed in the**BWOIExample**sample since the CMake-generated vcxproj uses the MSBuild rules for the Microsoft GDK. The Directory.Build.props file needs to be present and the environment properly setup when *generating* the CMake as well as building the resulting SLN/VCXPROJ.

When generating from the command line using BWOI, you can specify **GDK\_DXCTool** by adding -DGDK\_DXCTool=<path>, where <path> takes the form <path to GDK>\<edition number>\GXDK\bin\<XboxOne or Scarlett>\dxc.exe. For example:

-DGDK\_DXCTool="d:\xtrctd.sdks\BWOIExample\Microsoft GDK\210600\GXDK\bin\XboxOne\dxc.exe".

## 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",

}

],

If using CMake directly and the VS Generator while not using the Visual Studio integration, you can also specify this via **set\_property**.

set\_property(TARGET ${PROJECT\_NAME} PROPERTY VS\_GLOBAL\_ClearDevCommandPromptEnvVars "false")

set\_property(TARGET ${PROJECT\_NAME} PROPERTY VS\_GLOBAL\_VCToolsVersion "14.20.27508")

# Version History

|  |  |
| --- | --- |
| **April 2020** | Initial release of this version of CMake sample. |
| **June 2020** | Updated for the 2006 GDK FAL release. |
| **August 2020** | Add explicitly set VS startup project.  Updated with side-by-side details. |
| **November 2020** | Extension libraries require explicitly references to .lib files.  Add xmem.lib and xg\_\*.lib to link for Gaming.Xbox.\*.x64. |
| **February 2021** | Fix for Gaming.Desktop.x64 deployment. |
| **April 2021** | Add appnotify.lib to resolve link issues with Gaming.Desktop.x64.  Add LargeLogo.png. |
| **June 2021** | Fixed bug to explicitly set edition number in generated vcxproj for side-by-side scenarios.  General code cleanup. |
| **August 2021** | Improvements for toolchain files.  Additional notes for BWOI scenarios. |
| **October 2021** | Further notes for BWOI scenarios. |