



Read Me

Release 13.0

NVIDIA Corporation

Dec 16, 2025

Contents

1	Read Me	1
1.1	Release Notes	1
1.2	Sample Applications	2
1.3	System Requirements	8
1.4	Building Samples	14

Chapter 1. Read Me

1.1. Release Notes

What's new in Video Codec SDK v13.0?

Extended SDK support to NVIDIA Jetson Thor platform

Encode Features:

1. H264 interlaced, 10-bit and 4:2:2 encoding support on Blackwell GPUs
2. HEVC 4:2:2 encoding support on Blackwell GPUs
3. MV-HEVC: Support MultiView encoding in HEVC
4. AV1 LTR: Support Long-Term Reference frame in AV1
5. Support AV1 and HEVC Temporal Layer encoding
6. Support AV1 and HEVC MaxCLL, Mastering Display, and ITU-T T.35 SEI/Metadata

Decode Features:

1. 2x H264 throughput and maximum supported resolution 8192x8192 on Blackwell GPUs.
2. H264 High10/High422 Profile support (exclude MBAFF) on Blackwell GPUs
3. HEVC Main 4:2:2 10/12 profile support (exclude YUV400) on Blackwell GPUs
4. Dynamic decode surfaces allocation

Package Contents

This package contains the following:

1. Sample applications demonstrating various encoding/decoding/transcoding capabilities
 - ▶ [.\Samples\]
2. NVIDIA video encoder API header
 - ▶ [.\Interface\nvEncodeAPI.h]
3. NVIDIA video decoder API headers
 - ▶ [.\Interface\cuviddec.h]
 - ▶ [.\Interface\nvcuvid.h]

4. NVIDIA video decoder and encoder stub libraries

- ▶ [.\Lib\linux\stubs\x86_64\libnvcuvid.so]
- ▶ [.\Lib\linux\stubs\x86_64\libnvidia-encode.so]
- ▶ [.\Lib\linux\stubs\aaarch64\libnvcuvid.so]
- ▶ [.\Lib\linux\stubs\aaarch64\libnvidia-encode.so]
- ▶ [.\Lib\Win32\nvcuvid.lib]
- ▶ [.\Lib\Win32\nvencodeapi.lib]
- ▶ [.\Lib\x64\nvcuvid.lib]
- ▶ [.\Lib\x64\nvencodeapi.lib]

The sample applications provided in the package are for demonstration purposes only and may not be fully tuned for quality and performance. Hence the users are advised to do their independent evaluation for quality and/or performance.

1.2. Sample Applications

Video Codec SDK contains various sample applications that demonstrate how to use NVENC and NVDEC APIs. These applications are primarily developed as reference code for developers to understand API usage, quickly run experiments, and compare the control/data flow with their applications. These samples are built on top of reusable classes `NvEncoder` and `NvDecoder` which encapsulate the main functionality and provide a simple high-level programming interface for quick development. A few other applications are also included as samples, which demonstrate how to use NVIDIA codec APIs optimized for scalability along with a few advanced features for quality/performance tradeoff.

Folder Structure:

```
Samples
|--AppDecode
This folder contains all the decoder sample applications.
|--AppEncode
This folder contains all the encoder sample applications.
|--AppTranscode
This folder contains all the transcode sample applications.
|--External
This folder contains the external dependencies required to build samples
|--NvCodec
This folder contains classes that implement high-level interface for NVENC and NVDEC
  ↳ APIs.
|--Utils
This folder contains the utility code used by samples.
```

Help section is available for all applications. Execute any application with command line parameter `-h` or `--help` to get basic usage information. Execute with `-A` or `--advanced-options` to get detailed usage information. Help section will contain list of supported parameters indicating which ones are mandatory and which ones are optional. For optional parameters, it also mentions what default value is set in case user does not pass any value to them.

Here is a short description of what each application does:

Decoder Applications

AppDec

This sample application illustrates the demuxing and decoding of a media file followed by resize and crop of the output frames. The application supports 4:2:0, 4:2:2 and 4:4:4 chroma formats (planar and semi-planar), e.g., YUV420P/YUV420P16, YUV422P/YUV422P16, YUV444P/YUV444P16, as well as NV12 and P016 output formats.

Sample command line:

```
AppDec.exe -i input.h264 -o output.yuv -gpu 0
```

AppDecD3D

This sample application illustrates the decoding of media file and display of decoded frames in a window. This is done by CUDA interop with D3D (both D3D9 and D3D11).

Sample command line:

```
AppDecD3D.exe -i input.h264 -d3d 11
```

AppDecGL

This sample application illustrates the decoding of media file and display of decoded frames in a window. This is done by CUDA interop with OpenGL. Synchronization between rendering and decode thread is achieved using ConcurrentQueue implementation.

Sample command line:

```
AppDecGL.exe -i input.h264
```

AppDecImageProvider

This sample application illustrates the decoding output of a media file in a desired color format. The application supports native YUV and various RGB (bgra, bgrp, bgra64) output formats.

Output formats supported: NV12/P016 (native YUV), BGRA/BGRP/BGRA64 (RGB).

Sample command line:

```
AppDecImageProvider.exe -i input.h264 -o output.bgra
```

AppDecLowLatency

This sample application demonstrates low-latency decoding. This feature sets CUIDPARSER-PARAMS::ulMaxDisplayDelay = 0 while creating the parser object to get an output frame as soon as it is available for display, without any delay.

There is a command-line option “force_zero_latency” that allows capturing decoded frames immediately after decode. The feature will work for streams having I and P frames only (same display and decode order), as the application captures decoded frames just after decode in the decode callback function.

Sample command line:

```
AppDecLowLatency.exe -i input.h264 -force_zero_latency
```

AppDecMem

This sample application is similar to AppDec. It illustrates how to demux and decode media content from memory buffer. It allocates AVIOContext explicitly and also defines method to read data packets

from input file. For simplicity, this application reads the input stream and stores it in a buffer before invoking the demuxer.

Sample command line:

```
AppDecMem.exe -i input.h264
```

AppDecMultiFiles

This sample application illustrates the decoding of multiple files with/without using the decoder Re-configure API. It also displays the time taken for decoder creation and destruction. Multiple files are specified using `-filelist` commandline option. The app will decode files in a sequential manner.

Sample command line:

```
AppDecMultiFiles.exe -filelist files.txt
```

AppDecMultiInput

This sample application demonstrates how to decode multiple raw video files and post-process them with CUDA kernels on different CUDA streams. This sample applies Ripple effect as a part of post processing. The effect consists of ripples expanding across the surface of decoded frames.

Sample command line:

```
AppDecMultiInput.exe -i input1.yuv -s 1920x1080 -i input2.yuv -s 1280x720
```

AppDecPerf

This sample application measures decoding performance in FPS. The application creates multiple host threads and runs a different decoding session on each thread. The number of threads can be controlled by the CLI option `-thread`. The application creates 2 host threads, each with a separate decode session, by default. The application supports measuring the decode performance only (keeping decoded frames in device memory) as well as measuring the decode performance including transfer of frames to the host memory.

Sample command line:

```
AppDecPerf.exe -i input.h264 -thread 2
```

Encoder Applications

AppEncCuda

This sample application illustrates encoding of frames in CUDA device buffers. The application reads the image data from file and loads it to CUDA input buffers obtained from the encoder using `NvEncoder::GetNextInputFrame()`. The encoder subsequently maps the CUDA buffers for encoder using `NvEncodeAPI` and submits them to NVENC hardware for encoding as part of `EncodeFrame()` function. The NVENC hardware output is written in system memory for this case.

This sample application also illustrates the use of video memory buffer allocated by the application to get the NVENC hardware output. This feature can be used for H264 ME-only mode, H264 encode and HEVC encode. This application copies the NVENC output from video memory buffer to host memory buffer in order to dump to a file, but this is not needed if application chooses to use it in some other way.

Since encoding may involve CUDA pre-processing on the input and post-processing on output, use of CUDA streams is also illustrated to pipeline the CUDA pre-processing and post-processing tasks, for output in video memory case.

CUDA streams can be used for H.264 ME-only, HEVC ME-only, H264 encode, HEVC encode and AV1 encode.

Input formats supported: IYUV/YV12, NV12, P010, NV16, P210, YUV444, YUV444P16, BGRA, BGRA10, AYUV, ABGR, ABGR10.

Sample command line:

```
AppEncCuda.exe -i input.yuv -s 1920x1080 -if iyuv -o out.h264
```

AppEncD3D9

This sample application illustrates encoding of frames in IDirect3DSurface9 surfaces. There are 2 modes of operation demonstrated in this application. In the default mode application reads RGB data from file and copies it to D3D9 surfaces obtained from the encoder using `NvEncoder::GetNextInputFrame()` and the RGB surface is submitted to NVENC for encoding. In the second case (`-nv12` option) the application performs a color space conversion from RGB to NV12 using DXVA's `VideoProcessBlt` API call and the NV12 surface is submitted for encoding.

Input formats supported: RGB; NV12 via `-nv12` parameter.

Sample command line:

```
AppEncD3D9.exe -i input.bgra -s 1920x1080 -nv12 -o out.h264
```

AppEncD3D11

This sample application illustrates encoding of frames in ID3D11Texture2D textures. There are 2 modes of operation demonstrated in this application. In the default mode application reads RGB data from file and copies it to D3D11 textures obtained from the encoder using `NvEncoder::GetNextInputFrame()` and the RGB texture is submitted to NVENC for encoding. In the second case (`-nv12` option) the application converts RGB textures to NV12 textures using DXVA's `VideoProcessBlt` API call and the NV12 texture is submitted for encoding.

This sample application also illustrates the use of video memory buffer allocated by the application to get the NVENC hardware output. This feature can be used for H264 ME-only mode, H264 encode, HEVC encode and AV1 encode.

Input formats supported: RGB; NV12 via `-nv12` parameter.

Sample command line:

```
AppEncD3D11.exe -i input.bgra -s 1920x1080 -nv12 -o out.h264
```

AppEncD3D12

This sample application illustrates encoding of ID3D12Resource. This feature can be used for H264 encode, HEVC encode and AV1 encode.

Input formats supported: RGB.

Sample command line:

```
AppEncD3D12.exe -i input.bgra -s 1920x1080 -o out.h264
```

AppEncDec

This sample application illustrates the encoding and streaming of a video with one thread while another thread receives and decodes the video. HDR video streaming is also demonstrated in this application.

Input formats supported: IYUV, NV12, NV16, P010, P210, BGRA, BGRA64.

Sample command line:

```
AppEncDec.exe -i input.h264 -o out.h264
```

AppEncGL

This sample application illustrates encoding of frames stored in OpenGL textures. The application reads frames from the input file and uploads them to the textures obtained from the encoder using `NvEncoder::GetNextInputFrame()`. The encoder subsequently maps the textures for encoder using `NvEncodeAPI` and submits them to NVENC hardware for encoding as part of `NvEncoder::EncodeFrame()`.

The X server must be running and the `DISPLAY` environment variable must be set when attempting to run this application.

Input formats supported: IYUV, NV12.

Sample command line:

```
AppEncGL.exe -i input.yuv -s 1920x1080 -if iyuv -o out.h264
```

AppEncLowLatency

This sample application demonstrates low latency encoding features and other QOS features like bitrate change and resolution change. The application uses the CUDA interface to demonstrate the above features but can also be used with the D3D or OpenGL interfaces. There are 2 cases of operation demonstrated in this application, controlled by the CLI option `-case`. In the first case the application demonstrates bitrate change at runtime without the need to reset the encoder session. The application reduces the bitrate by half and then restores it to the original value after 100 frames. The second case demonstrates dynamic resolution change feature where the application can reduce resolution depending upon bandwidth requirement. In the application, the encode dimensions are reduced by half and restored to the original dimensions after 100 frames.

Input formats supported: IYUV, NV12, NV16, P210.

Sample command line:

```
AppEncLowLatency.exe -i input.yuv -s 1920x1080 -if iyuv -case 1 -o out.h264
```

AppEncME

This sample application illustrates the use of NVENC hardware to calculate motion vectors. The application uses the CUDA device type and associated buffers when demonstrating the usage of the ME-only mode but can be used with other device types like D3D and OpenGL.

Input formats supported: IYUV, NV12, YV12, YUV444, P010, YUV444P16, BGRA, ARGB10, AYUV, ABGR, ABGR10.

Sample command line:

```
AppEncME.exe -i input.yuv -s 1920x1080 -if iyuv
```

AppEncMultinstance

This sample application was created to accelerate file compression storage applications. It does this by splitting the input video into N separate and independent video portions, i.e., independent GOPs (Split GOP). After being encoded independently, the compressed video portions are then written to file preserving the original order generating a single output bitstream. More than one encoding session thread can be used to encode the several independent video portions. Using more than 1 encoding session threads should allow for speedups when using NVIDIA GPUs with more than 1 NVENC. The number of portions the input video should be partitioned in is controlled by the CLI option `-nf` and the number of encoding session threads `-thread`. Note that on systems with GeForce GPUs, the

number of simultaneous encode sessions allowed on the system is restricted to 5 sessions. There are separate threads for: 1. reading the RAW input frames from disk; 2. copying the RAW frames from RAM to VRAM, encoding and copying the compressed data from VRAM to RAM; 3. writing the compressed data to the output file. Additionally, the main thread is only used for initialization and to create work queues for the described threads.

Input formats supported: IYUV, NV12, YV12, YUV444, P010, YUV444P16, BGRA, ARGB10, AYUV, ABGR, ABGR10.

Sample command line:

```
AppEncMultiInstance.exe -i input.yuv -s 1920x1080 -if iyuv -nf 4 -thread 2 -o out.h264
```

AppEncPerf

This sample application measures encoding performance in FPS. The application creates multiple host threads and runs a different encoding session on each thread. The number of threads can be controlled by the CLI option `-thread`. The application creates 2 host threads, each with a separate encode session, by default. Note that on systems with GeForce GPUs, the number of simultaneous encode sessions allowed on the system is restricted to 3 sessions.

Input formats supported: IYUV, NV12, YV12, NV16, YUV444, P010, P210, YUV444P16, BGRA, ARGB10, AYUV, ABGR, ABGR10.

Sample command line:

```
AppEncPerf.exe -i input.yuv -s 1920x1080 -if iyuv -thread 2 -frame 2000
```

AppEncQual

This sample application demonstrates an Iterative Encoder implementation. A constant quality mode is implemented where the user is able to specify a minimum and maximum PSNR-Y as well as maximum number of iterations per frame. The Iterative Encoder will:

1. Interrupt the encoder state after each encoded frame;
2. Check the Reconstructed frame's PSNR-Y (Reconstructed Frame Output API);
3. Compare against the user defined range of desired PSNRs;
4. Adjust the QP/CQ parameter for the next iteration (Reconfigure API);
5. After the desired PSNR range or maximum number of iterations is reached, the encoder state is advanced and the next frame is encoded

This sample is compatible with rate controls Constant QP and VBR Constant Quality. The QP/CQ parameter is adjusted based on qpDelta input parameter (default: 1).

Input formats supported: IYUV, NV12, YV12, NV16, P210, YUV444.

Sample command line:

```
AppEncQual.exe -i input.yuv -s 1920x1080 -if iyuv -maxiter 3 -minpsnr 35 -maxpsnr 40
```

AppEncExternalMEHint

This sample application demonstrates passing external ME hints to NVENC. The application uses the CUDA interface to demonstrate the above feature but can also be used with the D3D or OpenGL interfaces. When external ME hints are enabled, the application expects configuration files that specify hint counts per block and paths to input hint files for each frame, allowing users to provide custom motion estimation guidance to the encoder.

Input formats supported: IYUV, NV12.

Sample command line:

```
AppEncExternalMEHint.exe -i input.yuv -s 1920x1080 -if nv12 -externalMEHintConfigFile  
↪ hints.cfg
```

AppMotionEstimationVkCuda

This sample application demonstrates feeding of CUarrays to EncodeAPI for the purposes of motion estimation between pairs of frames, using the H.264 motion estimation-only mode. The CUarrays registered with EncodeAPI have not been created by the application but have been obtained through the interop of CUDA with the Vulkan graphics API.

Transcode Applications

AppTranscode

This sample application demonstrates transcoding of an input video stream. If requested by the user, the bit-depth of the decoded content will be converted to the target bit-depth before encoding. The only supported conversions are from 8-bit to 10-bit (per component) and vice versa.

Sample command line:

```
AppTranscode.exe -i input.h264 -o out.h264
```

AppTransOneToN

This sample application demonstrates 1:N transcoding of a single input stream. Decoding of frames from the input stream takes place on the main thread and new threads are spawned for each output stream. A different resolution can be specified for each output stream and the decoded frames will be scaled as required. If no output resolutions are specified, this application will generate two streams: one of 1280x720 and the other of 800x480.

Sample command line:

```
AppTransOneToN.exe -i input.h264 -o out -r 1280x720 800x480
```

AppTransPerf

This sample application measures transcoding performance in FPS. This sample application takes a single input stream and spawns N pairs of threads. In each pair, one thread is responsible for decoding the input stream and making the decoded frames available to the other thread for encoding.

Sample command line:

```
AppTransPerf.exe -i input.h264 -thread 2
```

1.3. System Requirements

- ▶ GPU: NVIDIA Maxwell/Pascal/Volta/Turing/Ampere/Ada/Hopper/Blackwell GPU with hardware video accelerators
- ▶ Jetson: NVIDIA Jetson Thor with hardware video accelerators
 - ▶ Refer to the NVIDIA Video Codec SDK Support Matrix web page (<https://developer.nvidia.com/video-encode-and-decode-support-matrix>) for GPU and Jetson which support video encoding and decoding acceleration.

- ▶ Video Codec SDK can be downloaded from <https://developer.nvidia.com/nvidia-video-codec-sdk>
- ▶ For Jetson, the Video Codec SDK (and all required dependencies) can also be installed directly using APT by running the following command in device terminal:

```
sudo apt install nvidia-video-codec-sdk
```

- ▶ Video Codec SDK is available in GitLab at <https://gitlab.com/nvidia/video/video-codec-sdk>
- ▶ Documents can be browsed online at <https://docs.nvidia.com/video-technologies/video-codec-sdk/index.html>
- ▶ Windows
 - ▶ Driver version 570 and above
 - ▶ CUDA 11.0 or higher Toolkit
- ▶ Linux
 - ▶ Driver version 570 and above
 - ▶ CUDA 11.0 or higher Toolkit
- ▶ Jetson Linux
 - ▶ Version 38.4 and above
 - ▶ CUDA 13.0 or higher Toolkit
- ▶ Visual Studio Solution and Linux Makefiles can now be generated using CMake. CMake 3.9 or later is required for SDK 10.0 and higher. Self-extracting scripts or installers for CMake can be downloaded from <https://cmake.org/download/>.

Windows Configuration Requirements

- ▶ Vulkan SDK.

To build and run the AppMotionEstimationVkCuda sample application.

Actions	Overview
Get package	The Vulkan SDK needs to be installed to build and run the AppMotionEstimationVkCuda sample application. Vulkan SDK can be downloaded from https://vulkan.lunarg.com/sdk/home .
Set environment variable	VULKAN_SDK: pointing to Vulkan SDK install directory.

- ▶ FFMPEG:

If the SKIP_FFMPEG_DEPENDENCY CMake variable is set to TRUE, the following applications dependent on FFmpeg will not be generated: AppDec, AppDecD3D, AppDecGL, AppDecImageProvider, AppDecLowLatency, AppDecMem, AppDecMultiFiles, AppDecMultiInput, AppDecPerf, AppEncDec, AppTrans, AppTransOneToN and AppTransPerf.

Actions	Overview
Get package	Get FFMPEG LGPL shared build (version 7.1 or above) from BtbN repository .
Set cmake variable	During the CMake configuration phase (<i>Section 1.3</i>), set the FFMPEG_DIR CMake variable to point to the extracted FFmpeg directory containing the headers and libraries (i.e., the directory that contains the bin, lib, and include subdirectories).
	To set the FFMPEG_DIR CMake variable correctly during configuration, use the following syntax with an absolute or properly referenced path to the extracted FFmpeg directory: <code>-DFFMPEG_DIR=ffmpeg-master-latest-win64-gpl-shared/ffmpeg-master-latest-win64-gpl-shared</code> .
Skip library	If the user does not want to build any apps that depend on FFmpeg, they can set the CMake variable SKIP_FFMPEG_DEPENDENCY to TRUE to skip setting up FFmpeg libraries. This will exclude all applications that depend on FFmpeg. It can be set as: <code>-DSKIP_FFMPEG_DEPENDENCY=TRUE</code> .

► GLEW:

If the SKIP_GL_DEPENDENCY CMake variable is set to TRUE, AppDecGL will not be generated.

Actions	Overview
Get package	Get prebuilt GLEW binaries (version 2.1.0) from GLEW repository .
Set cmake variable	During the CMake configuration phase (<i>Section 1.3</i>), set the GLEW_DIR CMake variable to the extracted GLEW directory containing headers and libraries (i.e., the directory with bin, lib, and include subdirectories).
	To set the GLEW_DIR CMake variable correctly during configuration, use the following syntax with an absolute or properly referenced path to the extracted GLEW directory: <code>-DGLEW_DIR=glew-2.1.0-win32/glew-2.1.0</code> .
Skip library	If the user does not want to build any apps that depend on the GLEW library, they can set the CMake variable SKIP_GL_DEPENDENCY to TRUE to skip setting up the GLEW libraries. This excludes all GLEW-dependent applications. Use: <code>-DSKIP_GL_DEPENDENCY=TRUE</code> .

► FREEGLUT:

If the SKIP_GL_DEPENDENCY CMake variable is set to TRUE, AppDecGL will not be generated.

Actions	Overview
Get package	Get freeglut source code (version 3.4.0) from GLUT repository . The download link for version 3.4.0 can be found under the Stable releases section.

continues on next page

Table 4 – continued from previous page

Actions	Overview
Build libraries	Since freglut does not distribute prebuilt libraries, users need to build the libraries from the fetched source code. Detailed instructions for building the libraries from the freglut source code can be found in the README.cmake file in the freglut source directory. Ensure both Release and Debug versions of freglut are built, as both are required for building VideoCodecSDK apps.
	After a successful build the lib/Debug subdirectory in freglut build directory will have freglut_static.lib and freglutd.lib files, and the lib/Release subdirectory in freglut build directory will have freglut.lib and freglut_static.lib files.
Set cmake variable	During the CMake configuration phase (<i>Section 1.3</i>), set the GLUT_DIR CMake variable to point to the freglut build directory containing the locally built libraries (i.e., the directory that contains the bin and lib subdirectories).
	GLUT_DIR must point to the directory containing the bin and lib subdirectories (with locally built libraries). GLUT_INC must point to the directory containing the GL subdirectory, which includes the freglut headers. -DGLUT_DIR=/freglut-3.4.0/lib -DGLUT_INC=/freglut-3.4.0/include.
Skip library	If the user does not want to build any apps that depend on the freglut library, they can set the CMake variable SKIP_GL_DEPENDENCY to TRUE to skip setting up the freglut libraries. This will exclude all applications that depend on freglut. It can be set as: -DSKIP_GL_DEPENDENCY=TRUE.

► Agility SDK:

Visual Studio 2017 and above, Windows 20H1 or later is required for building and running AppEncD3D12 sample application.

Actions	Overview
Get Agility SDK	Get the Agility SDK (D3D12SDKVersion 606 or later) from: Agility SDK .
Set cmake variable	AGILITY_SDK_BIN: To point to the directory containing the D3D12Core.dll for the platform.
Set environment variable	DXSDK_DIR: pointing to the DirectX SDK root directory.
	AGILITY_SDK_VER: D3D12SDKVersion of the Agility SDK version used.
Application Dependencies.	On building AppEncD3D12, D3D12 directory which contains the required dlls from Agility SDK is created along with AppEncD3D12.exe. Make sure to copy the D3D12 directory along with AppEncD3D12.exe if the executable is moved to some other location.

► Plus all the requirements under [System Requirements](#) and *Common to all OS platforms*

Linux Configuration Requirements

- ▶ X11 and OpenGL, GLUT, GLEW libraries for video playback and display
- ▶ CUDA Toolkit is mandatory if client has Video Codec SDK 8.1 or above on his/her machine.

Actions	Overview
FFMPEG version	The sample applications have been compiled and tested against the libraries and headers from FFmpeg- 7.1.
Configure FFMPEG	While configuring FFmpeg on Linux, it is recommended not to use the <code>-disable-decoders</code> option. This configuration is known to cause a channel error (XID 31) when executing sample applications with certain clips or may result in unexpected behavior.
Build FFMPEG	Add the directory (default: <code>/usr/local/lib/pkgconfig</code>) to the <code>PKG_CONFIG_PATH</code> environment variable. This is required by the Makefile to determine the include paths for the FFmpeg headers. Add the directory where the FFmpeg libraries are installed to the <code>LD_LIBRARY_PATH</code> environment variable. This is required to resolve run-time dependencies on the FFmpeg libraries.

- ▶ Vulkan SDK.

Actions	Overview
Vulkan SDK	The Vulkan SDK needs to be installed to build and run the AppMotionEstimationVkCuda sample application.

- ▶ Plus all the requirements under *System Requirements* and *Common to all OS platforms*

Windows Subsystem for Linux (WSL) Configuration Requirements

- ▶ CUDA Toolkit is mandatory to use Video Codec SDK 8.1 and higher”.
- ▶ Add the directory `/usr/lib/wsl/lib` to the `PATH` environment variable if it is not already included. This is required to ensure the path for the WSL libraries is available.
- ▶ FFMPEG: Libraries and headers from the FFmpeg project which can be downloaded and installed using the distribution's package manager or compiled from source.

Actions	Overview
FFMPEG version	The sample applications have been compiled and tested against the libraries and headers from FFmpeg- 7.1.
Configure FFMPEG	While configuring FFmpeg on Linux, it is recommended not to use the <code>-disable-decoders</code> option. This configuration is known to cause a channel error (XID 31) when executing sample applications with certain clips or may result in unexpected behavior.
Build FFMPEG	Add the directory (default: <code>/usr/local/lib/pkgconfig</code>) to the <code>PKG_CONFIG_PATH</code> environment variable. This is required by the Makefile to determine the include paths for the FFmpeg headers. Add the directory where the FFmpeg libraries are installed to the <code>LD_LIBRARY_PATH</code> environment variable. This is required to resolve run-time dependencies on the FFmpeg libraries.

- Plus all the requirements under *System Requirements* and *Common to all OS platforms*

Jetson Linux Configuration Requirements

- CUDA Toolkit is mandatory on Jetson device.

Actions	Overview
FFMPEG version	The sample applications have been compiled and tested against the libraries and headers from FFmpeg- 7.1.
Configure FFMPEG	While configuring FFmpeg on Linux, it is recommended not to use the <code>-disable-decoders</code> option. This configuration is known to cause a channel error (XID 31) when executing sample applications with certain clips or may result in unexpected behavior.
Build FFMPEG	Add the directory (default: <code>/usr/local/lib/pkgconfig</code>) to the <code>PKG_CONFIG_PATH</code> environment variable. This is required by the Makefile to determine the include paths for the FFmpeg headers. Add the directory where the FFmpeg libraries are installed to the <code>LD_LIBRARY_PATH</code> environment variable. This is required to resolve run-time dependencies on the FFmpeg libraries.

- Plus all the requirements under *System Requirements* and *Common to all OS platforms*

Common to all OS platforms

- ▶ CUDA toolkit can be downloaded from <http://developer.nvidia.com/cuda/cuda-toolkit>

The CUDA Toolkit and the related environment variables are optional to install if the client has Video Codec SDK 8.0. However, it is mandatory if the client has Video Codec SDK 8.1 or above installed on their machine. For newer GPU architectures, ensure that the downloaded CUDA toolkit version includes support for your specific GPU generation, as older toolkit versions may not support the latest GPU architectures.

- ▶ Vulkan SDK can be downloaded from <https://vulkan.lunarg.com/sdk/home>. Alternatively, it can be installed by using the distribution's package manager.
- ▶ NVIDIA does not provide support for FFMPEG; therefore, it is the responsibility of end users and developers, to stay informed about any vulnerabilities or quality bugs reported against FFMPEG. Users are encouraged to refer to the official FFMPEG website and community forums for the latest updates, patches, and support related to FFMPEG binaries and act as they deem necessary.
- ▶ Stub libraries (libnvcuvid.so and libnvidia-encode.so)

Actions	Overview
Stub libraries	They have been included as part of the SDK package to aid in application development on systems where the NVIDIA driver has not been installed. The sample applications in the SDK will link against these stub libraries during the build process. However, users must ensure that the stub libraries are not referenced when running the sample applications. A driver compatible with this SDK must be installed for the sample applications to work correctly.

1.4. Building Samples

Video Codec SDK uses CMake for building the samples. To build the samples, follow these steps:

Windows:

1. Install all dependencies for Windows, as specified in *Windows Configuration Requirements*
2. Extract the contents of the SDK into a folder.
3. Create a subfolder named "build" in Video_Codec_SDK_x.y.z/Samples
4. Open a command prompt in the "build" folder and run the following command, depending upon the version of Visual Studio on your computer.

Visual Studio 2022:

```
cmake -G"Visual Studio 17 2022" -A"x64" -DCMAKE_BUILD_TYPE=Release -DCMAKE_
↳INSTALL_PREFIX=. . .
```

Visual Studio 2019:

```
cmake -G"Visual Studio 16 2019" -A"x64" -DCMAKE_BUILD_TYPE=Release -DCMAKE_
↳INSTALL_PREFIX=. ..
```

Visual Studio 2017:

```
cmake -G"Visual Studio 15 2017" -A"x64" -DCMAKE_BUILD_TYPE=Release -DCMAKE_
↳INSTALL_PREFIX=. ..
```

To build VideoCodecSDK applications having FFmpeg, freeglut or GLEW dependencies, make sure that required headers and libraries are setup as mentioned in *Windows Configuration Requirements*.

For AppEncD3D12 project to be generated, cmake variables AGILITY_SDK_BIN and AGILITY_SDK_VER has to be set as mentioned in *Windows Configuration Requirements*. Add the following options to the above commands to set this cmake variables:

```
-DAGILITY_SDK_BIN=<Path to Agility SDK folder containing D3D12Core.dll> -DAGILITY_
↳SDK_VER=<D3D12SDKVersion of Agility SDK>
```

AppEncD3D12 project will not be generated if the above options are omitted.

This command will generate the necessary Visual Studio project files in the “build” folder. You can open `NvCodec.sln` file in Visual Studio and build. Alternatively, following command can be used to build the solution:

```
cmake --build . --target install --config Release
```

The application binaries will be available in `Samples/build`. Please note that the applications are validated only for x64 platform.

Linux:

1. Install all dependencies for Linux, as specified in *Linux Configuration Requirements*.
2. Extract the contents of the SDK into a folder.
3. Create a subfolder named “build” in `Video_Codec_SDK_x.y.z/Samples`
4. Use the following command to build samples in release mode:

```
cmake -DCMAKE_BUILD_TYPE=Release ..
make
make install
```

This will build and install the binaries of the sample applications. The application binaries will be available in the folder `Samples/build`.

Windows Subsystem for Linux:

1. Install all dependencies for Windows Subsystem for Linux, as specified in *Windows Subsystem for Linux (WSL) Configuration Requirements*.
2. Follow the build and installation steps provided above for Linux. Applications using OpenGL and Vulkan will not be built.

Jetson Linux:

When SDK is installed using zip package:

1. Install all dependencies for Jetson Linux, as specified in *Jetson Linux Configuration Requirements*.
2. Extract the contents of the SDK into a folder.
3. Create a subfolder named “build” in Video_Codec_SDK_x.y.z/Samples
4. Use the following command to build samples in release mode:

```
cmake -DCMAKE_BUILD_TYPE=Release ..
make
make install
```

When SDK is installed using APT:

Note

In this case, all required dependencies for Jetson Linux will be installed using APT. The steps specified in *Jetson Linux Configuration Requirements* are not required.

1. The SDK files are copied to /opt/nvidia/video-codec-sdk/x.y.z path.
2. Create a folder named “build” in the system.
3. Use the following command to build samples in release mode:

```
cmake -DCMAKE_BUILD_TYPE=Release /opt/nvidia/video-codec-sdk/x.y.z/Samples
make
make install
```

This will build and install the binaries of the sample applications. The application binaries will be available in the created “build” folder. Applications using OpenGL will not be built.

Setting Up Cross-Platform Support for Jetson Linux:

This section explains how to set up the cross-compilation environment for building Samples on the host system. This section is applicable to Jetson Linux platform only.

Key terminology:

- **Host system** means the x86 based server where developers will build video codec SDK samples.
- **Jetson board** means the target board where developers will run video codec SDK samples.

Before proceeding with cross-compilation setup, ensure that video codec SDK samples can be built natively on the Jetson device without any issue. If a complete build environment has not been set up on the Jetson device, follow the build and installation steps provided in the *Jetson Linux* section above.

Now, the following steps should be executed on your host system:

1. Clone the target rootfs from Jetson device to the host system.

Follow the instructions in the section “Cloning rootfs with initrd” under Flashing Support for Jetson Thor in the Jetson Linux Driver Package Developer Guide:

<https://docs.nvidia.com/jetson/archives/r38.4/DeveloperGuide/SD/FlashingSupportJetsonThor.html>

This process generates two sparse copies of the image: one with the name you specified (e.g., system.img) and another with a .raw suffix (e.g., system.img.raw).

Mount the .raw image on the host system using the following commands:

```
cd $HOME
mkdir -p jetson
sudo mount -t ext4 system.img.raw jetson
```

2. Install cross-compilation tools on host system with the following commands:

```
sudo apt install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
```

3. Install CUDA Toolkit 13.0 on the host system. CUDA toolkit on host system must match or be compatible with Jetson’s CUDA version.

4. Build samples on host system and deploy on target

- a. Extract the contents of the SDK into a folder.
- b. Create a subfolder named “build” in Video_Codec_SDK_x.y.z/Samples
- c. Use the following commands to build samples in release mode:

```
cmake -DCMAKE_SYSROOT=$HOME/jetson \
      -DCMAKE_TOOLCHAIN_FILE=./aarch64-cc-toolchain.cmake \
      -DCMAKE_BUILD_TYPE=Release ..
make
make install
```

This will build and install the binaries of the sample applications. The application binaries will be available in the folder Samples/build.

- d. Copy the sample binaries to the Jetson device and run the applications on the target Jetson device.

Note

Layout for several structures in NVENCODE API header have been changed in this SDK. Users are therefore recommended to populate the structures accordingly.

Notices

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgment, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

Trademarks

NVIDIA, the NVIDIA logo, and cuBLAS, CUDA, CUDA Toolkit, cuDNN, DALI, DIGITS, DGX, DGX-1, DGX-2, DGX Station, DLProf, GPU, Jetson, Kepler, Maxwell, NCCL, Nsight Compute, Nsight Systems, NVcaffe, NVIDIA Deep Learning SDK, NVIDIA Developer Program, NVIDIA GPU Cloud, NVLink, NVSHMEM, PerfWorks, Pascal, SDK Manager, Tegra, TensorRT, TensorRT Inference Server, Tesla, TF-TRT, Triton Inference Server, Turing, and Volta are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

©2022-2025, NVIDIA Corporation & affiliates. All rights reserved