NVIDIA CUDA VIDEO ENCODER

TB-06717-001_v5.5 | July 2013

Specification

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1. OVERVIEW

NVIDIA® CUDA Video Encoder is compliant with AVC/H.264 (MPEG-4 Part 10 AVC, ISO/IEC 14496-10). This encoder library is supported on all Tesla GPU and Fermi GPU architecture.

1.1. AVC Encoder

The H.264 encoder receives raw *YUV* frames and generates *NAL* packets. The encoder is developed in phases, with incremental tools/features support added at each phase. The final encoder design supports up to *High Profile @ Level 4.1*.

1.1.1. AVC Version 1.0

Goals

- Main Profile tools / features
- High Profile support
- Production quality encoder

Features

- Baseline/Main/High Profile. Up to Level 4.1
- Support B frames
- Configurable GOP
- HRD compliant for most encoded bit streams[1]

Availability:

- Included in NVIDIA GeForce® graphics drivers v181.20 for desktop PCs
- Supported on all CUDA-enabled GPUs with 32 scalar processor cores or more

[1] The rate control algorithm implements the HRD model but HRD compliance is not guaranteed for all settings and types of content.

1.1.2. AVC Version 1.1

Goals

- High Profile tools / features
- Interlaced encoding

Features

- High Profile. Up to Level 4.1
- Interlaced encoding (no *MBAFF/PicAFF*)
- Adaptive 8x8 / 4x4 transform
- *CBR* rate control

Availability:

- This will be introduced with the R185 graphics drivers in Q2'09
- Supported on all CUDA-enabled GPUs

  (V1.1 also extended the support to CUDA-enabled GPUs with less than 32 cores)

Chapter 2. ENCODER FEATURE CHECK LIST

Table 1 lists the supported features for the CUDA Encoder.

Table 1 AVC Encoder Features

Encoder Features	Version 1.0	Version 1.1
CAVLC	Y	Y
CABAC	Y	Y
Deblocking	Y	Y
Profile	Baseline, Main, High	Baseline, Main, High
Level	Up to 4.1	Up to 4.1
IDR Interval	Y	Y
I Interval	Y	Y
B between P	Y	Y
Interlaced	N	Y
Rate Control	CBR/Fixed QP	CBR/VBR/Fixed QP/VBR with Min QP
Max reference frames	1 (Fixed)	1 (Fixed)
ME search range configuration	N	N
Sub-pel refinement	Y	Y
PicAFF	N	N
Adaptive 8x8/4x4	N	Y

2.1. Encoder Presets

H.264 encoder Presets

Several encoder presets are provided that target specific requirements of certain encoding targets:

- iPod
- Sony PSP
- Blu-ray
- AVCHD

Encoding parameters are selected and tested on the targeted devices to make sure that encoded bit streams are compatible with the devices.

Chapter 3. ENCODER INTERFACES

NVIDIA CUDA Video Encoder exposes API through DirectShow filter interface or a C-library interface. Encoder availabilities for developers:

- Version 1.0 (R180 – Q1 09)
- Version 1.1 (R185 – Q2 09)

3.1. DIRECTSHOW

3.1.1. Filter

The NVIDIA CUDA Video encoder *DirectShow* filter supports GPU accelerated H264 and VC-1 video encoding. Table 2 lists the configuration of the filter interfaces.

Table 2 DirectShow Filter Interface Configuration

Filter Interfaces	**`IBaseFilter`**, **`ISpecifyPropertyPages`**, **`INVVESetting`**
Input pin media type	Supported types: ▸ MediaType: **`MEDIATYPE_VIDEO`** ▸ SubType: **`MEDIASUBTYPE_NULL`** ▸ Format: **`FORMAT_VideoInfo, FORMAT_VideoInfo2`** The following subtypes are accepted: ▸ **`MEDIASUBTYPE_YUY2`** ▸ **`MEDIASUBTYPE_IYUV`** ▸ **`MEDIASUBTYPE_UYVY`** ▸ **`MEDIASUBTYPE_YV12`**

	▸ **`MEDIASUBTYPE_NV12`**
Input Pin Interfaces	**`IMemInputPin`**, **`IPin`**, **`IQualityControl`**
Output pin media type	▸ MediaType: **`MEDIATYPE_Video`** ▸ SubType: **`MEDIASUBTYPE_H264`**
Output Pin Interfaces	**`IPin`**, **`IQualityControl`**
Filter CLSID	See DirectShow Filter GUIDS.
Interface ID	See DirectShow Filter GUIDS.
Property Page CLSID	Property page not exposed.
Executable	**`nvcuvenc.dll`**
Merit	**`MERIT_DO_NOT_USE`**

3.1.2. Visibility

The NVIDIA video encoder *DirectShow* filter visibility is limited. It cannot be used in the *GraphEdit* utility. However, it can be used in a test application using the *Filter GUIDs*.

3.1.3. Invvesetting Interface

DirectShow Filter *INVVESetting* interface can be used to get capability and set/get the encoder parameters. Table 3 lists the methods used by this interface.

Table 3 DirectShow Filter INVVESetting Interface Methods

Methods	Description
`IsSupportedCodec`	Query if the codec format is supported by the encoder
`IsSupportedCodecProfile`	Query if the profile for codec format is supported by the encoder
`SetCodecType`	Set encoder codec format
`GetCodecType`	Get the current encoding format
`IsSupportedParam`	Query if the parameter type is supported
`SetParamValue`	Set the value of the specified parameter type

Methods	Description
`GetParamValue`	Query the current value of the specified parameter type
`SetDefaultParam`	Applies default settings of the encoding format
`GetSPSPPS`	Fetches the buffer containing SPS and PPS

3.1.3.1. IsSupportedCodec

Description

Query if the codec format is supported by the encoder.

Syntax

HRESULT **`IsSupportedCodec`**(DWORD **`dwCodecType`**)

Parameter

`dwCodecType`

[in] Codec type support to query

Return Value

S_OK: The format is supported.

E_NOINTERFACE: The format is not supported.

E_FAIL: No CUDA capability present.

Remarks

Only **`NV_CODEC_TYPE_H264`** is supported.

3.1.3.2. IsSupportedCodecProfile

Description

Query if the profile for codec format is supported by the encoder.

Syntax

HRESULT **`IsSupportedCodecProfile`** (DWORD **`dwCodecType`**, DWORD **`dwProfileType`**)

Parameter

`dwCodecType`

[in] Codec type support to query

`dwProfileType`

[in] Codec profile support to query.

Return Value

S_OK: The profile is supported

E_NOINTERFACE: The profile is not supported

E_FAIL: No CUDA capability present

Remarks

For **`dwCodecType`**, only **`NV_CODEC_TYPE_H264`** is supported.

For **`dwProfileType`**, **`NVVE_H264_PROFILE_BASELINE`** and **`NVVE_H264_PROFILE_MAIN`** are supported for the H.264 codec.

`NVVE_H264_PROFILE_HIGH` support is limited (only at header bits in bitstream).

3.1.3.3. SetCodecType

Description

Set encoder codec format.

Syntax

HRESULT **`SetCodecType`**(DWORD **`dwCodecType`**).

Parameter

`dwCodecType`

[in] Codec format to be set

Return Value

S_OK: Sucessful

E_FAIL: Fail

Remarks

For **`dwCodecType`**, only **`NV_CODEC_TYPE_H264`** is supported.

This API must be called before the filter goes to run state, otherwise the graph will not play.

3.1.3.4. GetCodecType

Description

Get the current encoding format.

Syntax

HRESULT **`GetCodecType`**(DWORD **`*pdwCodecType`**)

Parameter

`pwdCodecType`

[out] Pointer to store the current encoding format

Return Value

S_OK: Successful.

E_FAIL: The encoding format is not initialized.

E_POINTER: **`pwdCodecType`** is NULL pointer.

Remarks

If successful, **`*pdwCodecType`** stores the current encoding format.

3.1.3.5. IsSupportedParam

Description

Query if the parameter type is supported.

Syntax

HRESULT **`IsSupportedParam`**(DWORD **`dwParamType`**)

Parameter

`dwParamType`

[in] Parameter support to query

Return Value

S_OK: The parameter is supported.

E_FAIL: The parameter is not supported.

Remarks

Parameter types are listed in Encoder Parameters.

3.1.3.6. SetParamValue

Description

Set the value of the specified parameter type. The **`pData`** points to a memory region storing the value of the parameter. The parameter can be a data structure, which must match the size of the parameter type.

Syntax

HRESULT **`SetParamValue`**(DWORD **`dwParamType`**, LPVOID **`pData`**)

Parameter

`dwParamType`

[in] Parameter to set

`pData`

[in] This pointer points to memory storing the value(s) of the parameter

Return Value

S_OK: Successful.

E_FAIL: Fail to set the value (e.g. encoder state does not allow).

E_NOTIMPL: Parameter is not adjustable.

E_UNEXPECTED: The encoding format is not initialized yet.

E_POINTER: **`pData`** is NULL pointer.

Remarks

Parameter types are listed in Encoder Parameters.

3.1.3.7. GetParamValue

Description

Query the current value of the specified parameter type.

Syntax

HRESULT **`GetParamValue`**(DWORD **`dwParamType`**, LPVOID **`pData`**)

Parameter

`dwParamType`

[in] Parameter to query

`pData`

[out] This pointer points to memory to store the value(s) of the parameter

Return Value

S_OK: Successful.

E_NOTIMPL: The parameter is not supported.

E_UNEXPECTED: The encoding format is not initialized.

E_POINTER: **`pData`** is NULL pointer.

Remarks

If querying is successful, **`*pData`** contains the current value of the parameter. Caller should guarantee that **`pData`** points to enough memory to store the data structure of the parameter.

3.1.3.8. SetDefaultParam

Description

Applies default settings of the encoding format.

Syntax

HRESULT **`SetDefaultParam(void)`**

Parameter

-

Return Value

S_OK: Successful.

E_UNEXPECTED: The encoding format is not set yet.

Remarks

Default values of parameters are mentioned in Encoder Parameters.

3.1.3.9. GetSPSPPS

Description

Fetches the buffer containing SPS and PPS.

Syntax

HRESULT c(unsigned char ***pSPSPPSbfr**, int **nSizeSPSPPSbfr**, int ***pDatasize**)

Parameter

pSPSPPSbfr

> [out] Pointer to the buffer for SPS and PPS. Memory for this buffer to be allocated by caller of this API.

nSizeSPSPPSbfr

> [in] Size in bytes of the buffer (**pSPSPPSbfr**)

pDatasize

> [out] Actual size in bytes of the buffer (**pSPSPPSbfr**)

Return Value

S_OK: Successful.

E_UNEXPECTED: The encoder is not initialized.

E_POINTER: NULL buffer pointer.

Remarks

Encoder should have been initialized prior to calling this API.

3.2. C-LIBRARY

3.2.1. API

The NVIDIA CUDA Video Encoder is also exposed as a C-library interface (API) to application. Following sections describe this API and different related structures in depth. Application programmer needs to be aware that the actual encoding APIs of this library have asynchronous operation to facilitate better utilization of CPU and GPU resources. The feedback mechanism to the application is through the callback functions that the application provides to the library at start of operation. See API Callback to find more details on callback mechanism.

3.2.2. API Function Interface

Table 4 C-library API Functions

Methods	Description
NvGetHWEncodeCaps	Query if the GPU supports the NVIDIA CUDA Video encoder.
NVCreateEncoder	Creates the NVIDIA CUDA Video Encoder library object for encoding.
NVIsSupportedCodec	Query if the codec format is supported.
NVIsSupportedCodecProfile	Query if the codec profile is supported.
NVSetCodec	Set the type of compression codec.
NVGetCodec	Get the type of compression codec.
NVIsSupportedParam	Query if the parameter type is supported.
NVSetParamValue	Set the value of the specified parameter type.
NVGetParamValue	Get the value of the specified parameter type.
NVSetDefaultParam	Applies default settings of the encoding format.
NVCreateHWEncoder	Allocate hardware resources for the encoder.
NVGetSPSPPS	Fetches the buffer containing SPS and PPS.
NVEncodeFrame	Encode one video picture.
NVRegisterCB	Register user defined callback functions to the encoder.
NVDestroyEncoder	Releases the NVIDIA encoder object.

3.2.2.1. NVGetHWEncodeCaps

Description

Query if the GPU supports the NVIDIA CUDA Video encoder.

Syntax

HRESULT **`__stdcall`** **`NVGetHWEncodeCaps`**(void).

Parameter

None

Return Value

S_OK: CUDA based encoding is supported

E_FAIL: No CUDA capability present

Remarks

None

3.2.2.2. NVCreateEncoder

Description

Creates the NVIDIA CUDA Video Encoder library object for encoding.

Syntax

HRESULT **`__stdcall`** **`NVCreateEncoder`**(NVEncoder **`*pNVEncoder`**)

Parameter

`pNVEncoder`

[out] This will have a valid encoder object handle on successful creation of encoder library instance.

Return Value

S_OK: Success. **`pNVEncoder`** parameter will have object handle.

E_OUTOFMEMORY: Not enough system memory.

Remarks

The object handle is returned though parameter **`pNVEncoder`**. Caller should not allocate any memory but just pass reference to **`NVEncoder`** type variable. This API does not commit the actual resources required for encoding.

3.2.2.3. NVIsSupportedCodec

Description

Query if the codec format is supported by the encoder.

Syntax

HRESULT **`__stdcall`** **`NVIsSupportedCodec`**(NVEncoder **`hNVEncoder`**, DWORD **`dwCodecType`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`dwCodecType`

[in] Codec type support to query

Return Value

S_OK: The format is supported

E_NOINTERFACE: The format is not supported

E_FAIL: No CUDA capability present

E_POINTER: Encoder handle is invalid

Remarks

Only **`NV_CODEC_TYPE_H264`** is are supported.

3.2.2.4. NVIsSupportedCodecProfile

Description

Query if the profile for codec format is supported by the encoder.

Syntax

HRESULT **`__stdcall`** **`NVIsSupportedCodecProfile`**(NVEncoder **`hNVEncoder`**, DWORD **`dwCodecType`**, DWORD **`dwProfileType`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`dwCodecType`

[in] Codec type support to query

`dwProfileType`

[in] Codec profile support to query

Return Value

S_OK: The profile is supported

E_NOINTERFACE: The profile is not supported

E_FAIL: No CUDA capability present

E_POINTER : Encoder handle is invalid

Remarks

For **`dwCodecType`**, only NV_CODEC_TYPE_H264 is supported.

For **`dwProfileType`**, NVVE_H264_PROFILE_BASELINE and NVVE_H264_PROFILE_MAIN are supported. NVVE_H264_PROFILE_HIGH support is limited (only at header bits in bitstream).

3.2.2.5. NVSetCodec

Description

Set encoder codec format.

Syntax

HRESULT **`__stdcall`** **`NVSetCodec`** (NVEncoder **`hNVEncoder`**, DWORD **`dwCodecType`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`dwCodecType`

[in] Codec format to be set

Return Value

S_OK: Successful

E_NOINTERFACE: Codec format is not supported

E_FAIL: No CUDA capability present

E_POINTER : Encoder handle is invalid

Remarks

For **`dwCodecType`**, only NV_CODEC_TYPE_H264 is supported.

3.2.2.6. NVGetCodec

Description

Get the current encoding format.

Syntax

HRESULT **`__stdcall`** **`NVGetCodec`** (NVEncoder **`hNVEncoder`**, DWORD **`*pdwCodecType`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`pdwCodecType`

[out] Pointer to store the current encoding format

Return Value

S_OK: Successful

E_FAIL: The encoding format is not initialized

E_POINTER: pdwCodecType is NULL pointer/ encoder handle is invalid

Remarks

If successful, **`*pdwCodecType`** stores the current encoding format.

3.2.2.7. NVIsSupportedParam

Description

Query if the parameter type is supported.

Syntax

HRESULT **`__stdcall`** **`NVIsSupportedParam`**(NVEncoder **`hNVEncoder`**, DWORD **`dwParamType`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`dwParamType`

[in] Parameter support to query

Return Value

S_OK: The parameter is supported

E_FAIL: The parameter is not supported

E_POINTER : Encoder handle is invalid

Remarks

Parameter types are listed in Encoder Parameters.

3.2.2.8. NVSetParamValue

Description

Set the value of the specified parameter type. The **`pData`** points to a memory region storing the value of the parameter. The parameter can be a data structure, which must match the size of the parameter type.

Syntax

HRESULT **`__stdcall`** **`NVSetParamValue`**(NVEncoder **`hNVEncoder`**, DWORD **`dwParamType`**, LPVOID **`pData`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`dwParamType`

[in] Parameter to set

`pData`

[in] This pointer points to memory storing the value(s) of the parameter

Return Value

S_OK: Successful.

E_FAIL: Fail to set the value (e.g. encoder state does not allow).

E_NOTIMPL: Parameter is not adjustable.

E_UNEXPECTED: The encoding format is not initialized yet.

E_POINTER: **`pData`** is NULL pointer/ encoder handle is invalid.

Remarks

Parameter types are listed in Encoder Parameters.

3.2.2.9. NVGetParamValue

Description

Query the current value of the specified parameter type.

Syntax

HRESULT **`__stdcall`** **`NVGetParamValue`**(NVEncoder **`hNVEncoder`**, DWORD **`dwParamType`**, LPVOID **`pData`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`dwParamType`

[in] Parameter to query

`pData`

[out] This pointer points to memory to store the value(s) of the parameter

Return Value

S_OK: Successful.

E_NOTIMPL: The parameter is not supported.

E_UNEXPECTED: The encoding format is not initialized.

E_POINTER: pData is NULL pointer/ encoder handle is invalid.

Remarks

If querying is successful, **`*pData`** contains the current value of the parameter. Caller should guarantee that **`pData`** points to enough memory to store the data structure of the parameter.

3.2.2.10. NVSetDefaultParam

Description

Applies default settings of the encoding format.

Syntax

HRESULT **`__stdcall`** **`NVSetDefaultParam`**(NVEncoder **`hNVEncoder`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

Return Value

S_OK: Successful.

E_UNEXPECTED: The encoding format is not set yet.

E_POINTER : Encoder handle is invalid.

Remarks

Default values of parameters are mentioned in Encoder Parameters.

3.2.2.11. NVCreateHWEncoder

Description

Allocate hardware resources for the encoder.

Syntax

HRESULT __stdcall **`NVCreateHWEncoder`**(NVEncoder **`hNVEncoder`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

Return Value

S_OK: Successful.

E_FAIL : Failed to allocate all hardware resources for NVIDIA CUDA video encoder.

E_POINTER : Encoder handle is invalid.

Remarks

None.

3.2.2.12. NVGetSPSPPS

Description

Fetches the buffer containing SPS and PPS.

Syntax

HRESULT **__stdcall** **NVGetSPSPPS**(NVEncoder **hNVEncoder** ,unsigned char ***pSPSPPSbfr**, int **nSizeSPSPPSbfr**, int ***pDatasize**)

Parameter

hNVEncoder

[in] Handle to the encoder instance

pSPSPPSbfr

[out] Pointer to the buffer for SPS and PPS. Memory for this buffer to be allocated by caller of this API.

nSizeSPSPPSbfr

[in] Size in bytes of the buffer (pSPSPPSbfr)

pDatasize

[out] Actual size in bytes of the buffer (pSPSPPSbfr)

Return Value

S_OK: Successful

E_UNEXPECTED: The encoder is not initialized

E_POINTER: NULL buffer pointer/encoder handle is invalid

Remarks

Encoder should have been initialized prior to calling this API.

3.2.2.13. NVEncodeFrame

Description

Encode one video picture.

Syntax

HRESULT **__stdcall** **NVEncodeFrame**(NVEncoder **hNVEncoder**, NVVE_EncodeFrameParams ***pFrmIn**, DWORD flag, **void *pData**)

Parameter

hNVEncoder

[in] Handle to the encoder instance

`pFrmIn`

[in]Various params for encoding the frame. See

`NVVE_EncodeFrameParams` structure flag

[in] (H.264 only) Contain the instruction for the encoding operations. The value of flag presently could be anyone of: **`FORCE_IDR`** = 0x04 (to force an IDR), **`FORCE_INTRA`** (to force an intra frame), **`INSERT_SPS`** (to insert sps), **`INSERT_PPS`** (to insert PPS). Note: Currently only **`FORCE_IDR`** is supported.

`pData`

[in] Pointer to data structure associated with the control instruction flag (if any required) or pointer to device memory input frame (**`CUdeviceptr`** type casted to void*) in case of **`NVVE_DEVICE_MEMORY_INPUT`**. When using **`NVVE_DEVICE_MEMORY_INPUT`** the *picBuf* element in the structure must be set to NULL **`pFrmIn.picBuf`** = NULL;

Return Value

S_OK: Successful

E_FAIL: Encoding has failed

E _ POINTER: Encoder handle is invalid

Remarks

`NVVE_EncodeFrameParams` contains the information about the incoming source pictures. Caller can control the encoding operation through flag and pData parameters.

3.2.2.14. NVRegisterCB

Description

Register user defined callback functions to the encoder

Syntax

void **`__stdcall NVRegisterCB`**(NVEncoder **`hNVEncoder`**, **`NVVE_CallbackParams cb`**, **`void *pUserdata`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

`cb`

[in] Structure containing the function pointers to a callback

`pUserdata`

[in] A void pointer which will be stored and passed back in the callbacks

Return Value

None

Remarks

The callback functions are called by the encoder to indicate the start of encoding a frame, end of encoding a frame, get output bitstream buffers and release output bitstream buffer. Also the encoder will receive and store a void **`*pUserdata`** which it will pass back in the callback functions.

3.2.2.15. NVDestroyEncoder

Description

Releases the NVIDIA CUDA Video Encoder library object.

Syntax

HRESULT **`__ stdcall`** **`NVDestroyEncoder`**(NVEncoder **`hNVEncoder`**)

Parameter

`hNVEncoder`

[in] Handle to the encoder instance

Return Value

S_OK: Successful

E _ POINTER: Encoder handle is invalid

Remarks

None

3.2.3. API Usage

This section provides an overview of the API usage in a typical encoding scenario.

A typical encoding session works as follows:

- The application queries whether the GPU supports the NVIDIA CUDA Video Encoder using **`NVGetHWEncodeCaps()`**.

- If the GPU supports the NVIDIA CUDA Video Encoder, the application sets up an encoding session by creating an encoder object using **`NVCreateEncoder()`**.
- The application queries what codecs are supported using **`NVIsSupportedCodec()`**.The application may further query to know if a particular profile for this codec is supported using **`NVISSupportedCodecProfile()`**.
- The application sets the desired codec using **`NVSetCodec()`**.
- The application may now want to set the different encoding parameters using **`NVSetParamValue()`** or it might choose to accept the default parameter settings using **`NVSetDefaultParam()`**. Default values for all parameters are described in sections below.
- If the application wants to query the parameters or encoding type, it can be done using **`NVGetParamValue()`** and **`NVGetCodec()`**.
- The application should register the callback handlers using **`NVRegisterCB()`**. The application may pass a userdata pointer here for later use. This can be called after **`NVCreateEncoder()`** and needs to be called before calling **`NVEncodeFrame()`**.
- The application allocates/commits hardware resources (for the codec type set earlier) using **`NVCreateHWEncoder()`**.
- To initiate encoding of a single frame the application calls **`NVEncodeFrame()`**.
- The callbacks work as follows:
  - On beginning the encode of a picture a callback to **`OnBeginFrame`** will be received.
  - The application needs to allocate memory for writing the encoded bitstream and pass it in **`AcquireBitstream`**.
  - The pointer to the encoded bitstream along with its size in bytes will be passed back in **`ReleaseBitstream`**. The application can use this pointer to store the encoded bitstream.
  - After the encoding for a picture is complete **`OnEndFrame`** will be called.
  - For detailed description of this mechanism please refer to API Callback.
- For the last picture to be encoded the application should set the bLast field of NVVE_EncodeFrameParams to true while calling **`NVEncodeFrame()`**.
- After encoding is complete the application should release the encoder object by calling **`NVDestroyEncoder()`**.

Pseudo code based on above description:

```
Main ()
{
    NVGetHWEncodeCaps() // checks CUDA encoding capabilities
    NVCreateEncoder()   // creates the encoder object
    NVSetCodec()        // set the codec type
    NVSetDefaultParam() // set up the default parameters
                        // allocate callback function pointers
                        // and any other resources that may be required
    NVRegisterCB()      // setup callbacks
    NVCreateHWEncoder() // allocate CUDA resources
    while (frames) {
        NVEncodeFrame()
        …
    }
    NVDestroyEncoder()  // destroys the encoder object
                        // clean up all the resources before quitting
}
```

```
// Callbacks
AcquireBitstream(int *pBufferSize, void *pUserdata)
{
    // specify size in *pBufferSize
    // return bitstream buffer;
}
ReleaseBitstream(int nBytesInBuffer, unsigned char *cb, void *pUserdata)
{
    // encoded bitstream for the current picture is returned in the
    // buffer cb points to
}
```

3.2.4. API Callback

Caller applications implement callback functions and register to the encoder using **`NVRegisterCB()`** function. These callback functions are used to acquire/release input frames and bitstream buffers. The application should be sending a void **`*pUserdata`** while calling **`NVRegisterCB()`** to which will be stored in the encoder dll and later on passed back on through the callback functions.

The callback order received will be in the following order:

`OnBeginFrame`#**`AcquireBitstream`**# **`ReleaseBitstream`**#**`OnEndFrame`**.

Alternatively, the caller can choose not to use the **`OnBeginFrame`** and **`OnEndFrame`** function pointers. If they set these two function pointers to NULL then only the callbacks to **`AcquireBitstream`** and **`ReleaseBitstream`** will be received.

The application allocate *sa* buffer to store the coded bitstream and pass it on to **`AcquireBitstream`**. **`pBufferSize`** points to the size of the buffer allocated. If the buffer allocated is not sufficient in size to contain the entire picture, **`AcquireBitstream`** and **`ReleaseBitstream`** are called multiple times.

The pointer to the buffer acquired in **`AcquireBitstream`** is stored and passed back on a **`ReleaseBitstream`** callback to the application as the 2nd argument to **`ReleaseBitstream`** (an unsigned char *).

The **`NVVE_BeginFrameInfo`** and NVVE_**`EndFrameInfo`** structures have two members each:

- **`nFrameNumber`**: zero-based frame number in display order (same for both fields of a frame)
- **`nPicType`**: this signifies the encoded picture type. It will take one of the values of **`NVVE_PIC_TYPE_IFRAME`**, **`NVVE_PIC_TYPE_PFRAME`** and **`NVVE_PIC_TYPE_BFRAME`**.

Chapter 4. ENCODER PARAMETERS

The encoder parameters can be configured using the *DirectShow Filter* **`INVVESetting`** interface or C -library API parameter configuration methods.

Table 5 lists encoder setup parameters.

Table 5 Encoder Setup Parameters

Parameter
`NVVE_OUT_SIZE` Specify the targeted encoding frame size. (use {0,0} for same size as detected at input) **Type:** INT[2] **Range:** Depends on profile level **Default:** {0, 0}
`NVVE_IN_SIZE` Specify the input picture dimension. (INT[0]:Width, INT[1]:Height) **Type:** For Dshow: NA (Query Parameter) **Range:** NA **Default:** NA **Type:** For C-lib: INT[2] **Range:** + ve integer **Default:** {0, 0}
`NVVE_ASPECT_RATIO` Specify the display aspect ratio. Encoder does not perform aspect ratio conversion. This should match the display aspect ratio of the input. **Type:** For Dshow: **`FLOAT`** (if not custom) **`NVVE_AspectRatio Params*`** (if custom) To specify width, height, type (**`DAR/SAR`**)

Parameter
Range: For Dshow: 4.0f/3.0f, 16.0f/9.0f, 1 . 0 f, custom **Default:** For Dshow: 4 . 0 f/3.0f **Type:** For C-lib: INT[3] {width, height, aspect_ ratio_type} Aspect_ratio_ type is of the type enum `NVVE_ASPECT_ RATIO_TYPE (DAR/SAR)*` **Range:** For C-lib: Array members should be integers >=0 For SAR, width and height values should be unsigned 16 bit integers as per ISO14496- 10 **Default:** For C-lib: {4, 3, 0}, correspondin g to 4:3 `DAR`
`NVVE_FIELD_ENC_MODE` Specify if the frame or field encoding mode is used (H.264 only). **Type:** `NVVE_FIELD _MODE*` **Range:** `MODE_FRAME, MODE_TOP_FIELD_ FIRST, MODE_BOTTOM_FIELD_FIRST` **Default:** `MODE_F RAME`
`NVVE_P_INTERVAL` This sets the distance of one P picture from the previous P picture. e.g. for `IBBPBBP`, set the value as 3 (H.264 only). **Type:** INT **Range:** 1-17 **Default:** 1
`NVVE_IDR_PERIOD` This is the IDR period for H264. **Type:** INT **Range:** +ve integer > = 1 **Default:** 15
`NVVE_DYNAMIC_GOP` The GOP structure is determined dynamically by the encoder (H.264 only). (does not take effect in V1.0) **Type:** INT **Range:** 0: disable 1: enable **Default:** 0
`NVVE_RC_TYPE` The rate control type. **Type:** `NVVE_RC _TYPE*` **Range:** For H.264: `RC_CQP, RC_VBR`, `RC_CBR,RC_VBR_MINQP`. For VC-1: `RC_CBR` **Default:** `RC_VBR` (H.264) `RC_CBR` (VC-1)

Parameter
NVVE_AVG_BITRATE The average bit rate in bps is the target bit rate used for VBR rate control. **Type:** INT **Range:** + ve integer **Default:** 6000000
NVVE_PEAK_BITRATE The maximum bit rate in bps is the peak bit rate used for VBR rate control. **Type:** INT **Range:** + ve integer **Default:** 6200000
NVVE_QP_LEVEL_INTER_P The QP level for inter P pictures. [Note: The QP will be clipped by the encoder if it exceeds the supported QP range]. For **RC_VBR_MINQP** rate control mode this parameter to be interpreted as min QP for inter P pictures (H.264 only). **Type:** INT **Range:** +ve integer, (0: default) **Default:** 28
NVVE_QP_LEVEL_INTER_B The QP level for inter B pictures. [Note: The QP will be clipped by the encoder if it exceeds the supported QP range]. For **RC_VBR_MINQP**rate control mode this parameter to be interpreted as min QP for inter B pictures (H.264 only). **Type:** INT **Range:** +ve integer, (0: default) **Default:** 28
NVVE_QP_LEVEL_INTRA The QP level for intra pictures. [Note: The QP will be clipped by the encoder if it exceeds the supported QP range]. For **RC_VBR_MINQP**rate control mode this parameter to be interpreted as min QP for intra pictures (H.264 only). **Type:** INT **Range:** +ve integer, (0: default) **Default:** 28
NVVE_FRAME_RATE Output frame rate (should be same as input frame rate). No frame rate conversion is performed if the output frame rate is not the same as the input frame rate.

Parameter
Type: For DShow: NVVEFrame Rate*If **NVVE_FRAME_RATE_NUMDEN**, then specify using **NVVE_FrameRate Descriptor** * **Range:** For DShow: as per enum. If **NVVE_FRAME_RATE_NUMDEN**, then Numerator >=0Denominator >0 **Default:** For DShow: 29.97 **Type:** For C-lib: INT[2] {numerator, denominator} **Range:** For C-lib: Numerator >=0. Denominator >0 **Default:** For C-lib: {30000, 1001}
NVVE_DEBLOCK_MODE Enable or disable de-blocking mode. This is only valid for H.264 (H.264 only). **Type:** INT **Range:** 0: disable, 1: enable **Default:** 1
NVVE_PROFILE_LEVEL Set the profile and level information. Level Setting: Other encoding parameters should be conformant to the level to avoid later failure at initialization. ‣ **Type:** INT ‣ **Range:** For H.264 Byte 0: 0x42:Baseline, 0x4d:Main, 0x64:High Byte 1: 0xff:auto select level. 10(0x0a), 11(0x0b), 12(0x0c), 13(0x0d),20(0x14), 21(0x15), 22(0x16), 30(0x1e), 31(0x1f), 32(0x20), 40(0x28), 41(0x29),42(0x2a), 50(0x32), 51(0x33): For Level 1.0, 1.1, 1.2, 1.3, 2.0,2.1, 2.2, 3.0, 3.1, 3.2,4.0, 4.1, 4.2, 5.0, 5.1 Byte2,3: reserved ‣ **Default:** 0xff42
NVVE_FORCE_INTRA Force generation of an intra frame (H.264 only). **Type:** INT **Range:** 1 **Default:** NA
NVVE_FORCE_IDR Force generation of an IDR (H.264 only). **Type:** INT **Range:** 1 **Default:** NA

Parameter
NVVE_CLEAR _STAT Clear the statistics values (H.264 only). **Type:** INT **Range:** 1 **Default:** NA
NVVE_SET_DEINTERLACE Set the deinterlace algorithm (H.264 only). **Type:** `NVVE_DI_MODE*` **Range:** `DI_OFF, DI_MEDIAN` **Default:** `DI_MEDIAN`
NVVE_PRESETS Set the encoding parameters according to the presets required for supported encoding targets (H.264 only). **Type:** `NVVE_ PRESETS _TARGET*` **Range:** `PSP, iPOD, AVCHD, BD, HDV_1440` **Default:** NA
NVVE_DISABLE_CA BAC Enable or disable `CABAC` (H.264 only). **Type:** INT **Range:** 0: enable, 1: disable **Default:** 0
NVVE_CONFIGURE_NALU_FRAMING_TYPE Configures the NAL unit framing type (H.264 only). **Type:** INT **Range:** 0: 0: start codes. 1, 2, 4: length prefixed NAL units of size 1, 2, or 4 bytes **Default:** 0
NVVE_DISABLE_SPS_PPS Enable or disable including sequence parameter set/picture parameter set (`SPS/PPS`) information in bitstream (H.264 only). **Type:** INT **Range:** 0: enable, 1: disable **Default:** 0

Parameter
`NVVE_SLICE_COUNT` Sets the number of slices per picture. Setting this to non-zero value will set the slice number per picture. If it is set to zero, the encoder will use its own default settings (H.264 only). Recommended settings for different output resolutions: <= 400x256 : 1 >= 400x256 and < 640x480 : 2 > 640x480 : 4 **Type:** INT **Range:** >=0 **Default:** 0, (decided by encoder)
`NVVE_GPU_OFFLOAD_LEVEL` Sets the GPU offload level. Applicable only to select GPUs and Codec. **Type:** **`NVVE_GPUOffloadLevel`** **Range:** Default, Estimators, All **Default:** Default
`NVVE_MULTI_GPU` Consider multi GPU usage if found suitable for the platform and codec. **Type:** INT **Range:** 1: consider, 0: don't consider **Default:** 1: consider
`NVVE_FORCE_GPU_SELECTION` Force encoding on a particular GPU in the system. **Type:** INT **Range:** -1: default, n: GPU ordinal number **Default:** - 1: default
`NVVE_DEVICE_MEMORY_INPUT` Input frame is provided to encoder in device memory. For Dshow: The input frame CUDA device memory pointer (**`CUdeviceptr`**) should be passed in the data pointer (type casted) of the input sample to the filter. For Clib: The input frame CUDA device memory pointer (**`CUdeviceptr`**) should be passed in the **`pData`** pointer (type casted) of the **`NVEncodeFrame()`** API. In this case the buffer pointer in **`NVVE_EncodeFrameParams`** is ignored **Type:** INT

Parameter
Range: 0 : system memory input, 1: device memory input **Default:** 0: system memory input
NVVE_DEVICE_CTX _LOCK Provide video context lock for device memory input. **Type:** `CUvideoctxlock` **Range:** Handle to be obtained from `NVCUVID APIs` **Default:** NA

For datatypes, see Invvesetting Interface/C-Library API Datatypes.

4.1. ENCODER QUERY PARAMETERS

Table 6 lists the encoder query parameters .

Table 6 Encoder Query Parameters

Parameter	Description	Type
NVVE_IN_SIZE	Get the input picture dimension. (INT[0]:Width, INT[1]:Height)	INT[2]
NVVE_STAT_NUM_CODED_FRAMES	Get the number of encoded frames so far.	LONGLONG
NVVE_STAT_NUM_RECEIVED_FRAMES	Get the number of received frames from input pin.	LONGLONG
NVVE_STAT_BITRATE	Get generated average bit rate in bps.	INT
NVVE_STAT_NUM_BITS_GENERATED	Number of bits generated.	LONGLONG
NVVE_GET_PTS_DIFF_TIME	Get the PTS difference between the last received sample and the current output PTS.	LONGLONG
NVVE_GET_PTS_CODED_TIME	Get the encoded PTS of the current frame.	LONGLONG
NVVE_GET_PTS_RECEIVED_TIME	Get the received PTS of the current frame.	LONGLONG
NVVE_STAT_ELAPSED_TIME	Get the elapsed time from the first received sample to the last received sample (in unit of 10000 ms).	LONGLONG

Parameter	Description	Type
`NVVE_STAT_QBUF_FULLNESS`	Get the number of samples queued at the input.	INT
`NVVE_STAT_PERF_FPS`	Get the runtime average of encoded frames per second. (considers only the time taken to start coding a frame and end coding a frame).	Float
`NVVE_STAT_PERF_AVG_TIME`	Get the average encoding time per frame (unit of 10 ms) (considers only the time taken to start coding a frame and end coding a frame).	DWORD
`NVVE_GPU_OFFLOAD_LEVEL_MAX`	Query maximum supported offload level for platform.	`NVVE_GPUOffloadLevel`
`NVVE_GET_GPU_COUNT`	Get count of capable GPUs.	INT
`NVVE_GET_GPU_ATTRIBUTES`	Get attributes of a particular GPU in the system. (provide GPU ordinal number).	`NVVE_GPUAttributes`

4.2. ENCODER PARAMETER DEPENDENCY

NVVE_RC_TYPE:

- For **RC_VBR**, the parameters **NVVE_AVG_BITRATE** and **NVVE_PEAK_BITRATE** take effect.
- For **RC_CQP**, the parameters **NVVE_QP_LEVEL_INTER_P**, **NVVE_QP_LEVEL_INTER_B** and **NVVE_QP_LEVEL_INTRA** take effect.
- For **RC_CBR**, the parameter **NVVE_AVG_BITRATE** takes effect.
- For **RC_VBR_MINQP**, the parameters **NVVE_AVG_BITRATE**, **NVVE_PEAK_BITRATE**, **NVVE_QP_LEVEL_INTER_P**, **NVVE_QP_LEVEL_INTER_B** and **NVVE_QP_LEVEL_INTRA** take effect.

  In this mode, since the encoder is limiting the min value of QP, the resulting bitrate can be lower – and potentially significantly lower – than the average bitrate.

NVVE_CLEAR_STAT:

Resets the statistic values for following parameters (mentioned in ENCODER QUERY PARAMETERS.

- **NVVE_STAT_NUM_CODED_FRAMES**, **NVVE_STAT_NUM_RECEIVED_FRAMES**,
- **NVVE_STAT_BITRATE**, **NVVE_STAT_NUM_BITS_GENERATED**,
- **NVVE_GET_PTS_DIFF_TIME**, **NVVE_GET_PTS_CODED_TIME**,

- **`NVVE_GET_PTS_RECEIVED_TIME`**, **`NVVE_STAT_ELAPSED_TIME`**,
- **`NVVE_STAT_QBUF_FULLNESS`**, **`NVVE_STAT_PERF_FPS`**,
- **`NVVE_STAT_PERF_AVG_TIME`**

NVVE_OUT_SIZE/NVVE_IN_SIZE:

For *DirectShow Filter*, **`NVVE_IN_SIZE`** will return the dimensions based on the pin connection at the input pin. For *C-lib* API, **`NVVE_IN_SIZE`** will set the input dimensions for the encoder. **`NVVE_OUT_SIZE`** is used to specify the targeted encoded output dimensions.

NVVE_DEVICE_MEMORY_INPUT / NVVE_DEVICE_CTX_LOCK:

Device Context Lock parameter must also be set if device memory input is enabled. Context lock should be created from **`cuvidCtxLockCreate`** API available in *NVCUVID*.

Appendix A. DIRECTSHOW FILTER GUIDS

A.1. Directshow Filter GUID

```
// {B63E31D0-87B5-477f-B224-4A35B6BECED6} 'Dshow NVIDIA Video Encoder Filter'
DEFINE_GUID(CLSID_NVIDIA_VideoEncoderFilter, 0xb63e31d0, 0x87b5, 0x477f, 0xb2,
            0x24, 0x4a, 0x35, 0xb6, 0xbe, 0xce, 0xd6);
```

A.2. Directshow Filter Invvesetting Interface GUID

```
// {4597F768-F60-4E5B-B697-67EB2614DCB5} ‘INVVESetting interface’
DEFINE_GUID(IID_INVVESetting, 0x4597f768, 0xf60, 0x4e5b, 0xb6, 0x97, 0x67,
            0xeb, 0x26, 0x14, 0xdc, 0xb5);
```

Appendix B. INVVESETTING INTERFACE/C-LIBRARY API DATATYPES

```
//
// Datatypes for DirectShow Filter INVVESetting Interface/C-library API to the
// video encoder
//

// Codec Type
// Used in IsSupportedCodec, IsSupportedCodecProfile, SetCodecType,
// GetCodecType interface functions
#define NV_CODEC_TYPE_MPEG1                   1 // not supported
#define NV_CODEC_TYPE_MPEG2                   2 // not supported
#define NV_CODEC_TYPE_MPEG4                   3 // not supported
#define NV_CODEC_TYPE_H264                    4

// Codec Profile Type
// Used in IsSupportedCodecProfile interface functions
#define NVVE_MPEG2_PROFILE_MAIN               0 // not supported
#define NVVE_H264_PROFILE_BASELINE            1
#define NVVE_H264_PROFILE_MAIN                2
#define NVVE_H264_PROFILE_HIGH                3

// Coded Picture Type                           // C-lib only
// Used in NVVE_BeginFrameInfo, NVVE_EndFrameInfo
#define NVVE_PIC_TYPE_IFRAME                  1
#define NVVE_PIC_TYPE_PFRAME                  2
#define NVVE_PIC_TYPE_BFRAME                  3

// Encoding Parameters
// Used in SetParamValue, GetParamValue interface functions enum EncodeParams
{
    NVVE_OUT_SIZE
    NVVE_ASPECT_RATIO,
    NVVE_FIELD_ENC_MODE,
    NVVE_P_INTERVAL,
    NVVE_IDR_PERIOD,
    NVVE_DYNAMIC_GOP,
    NVVE_RC_TYPE,
    NVVE_AVG_BITRATE,
    NVVE_PEAK_BITRATE,
    NVVE_QP_LEVEL_INTRA,
    NVVE_QP_LEVEL_INTER_P,
    NVVE_QP_LEVEL_INTER_B,
    NVVE_FRAME_RATE,
    NVVE_DEBLOCK_MODE,
    NVVE_PROFILE_LEVEL,
    NVVE_FORCE_INTRA,                 // DShow only
    NVVE_FORCE_IDR,                   // DShow only
    NVVE_CLEAR_STAT,                  // DShow only
    NVVE_SET_DEINTERLACE,
    NVVE_PRESETS, NVVE_ IN_SIZE,
    NVVE_STAT_NUM_CODED_FRAMES,       // DShow only
    NVVE_STAT_NUM_RECEIVED_FRAMES,    // DShow only
    NVVE_STAT_BITRATE,                // DShow only
    NVVE_STAT_NUM_BITS_GENERATED,     // DShow only
    NVVE_GET_PTS_DIFF_TIME,           // DShow only
    NVVE_GET_PTS_BASE_TIME,           // DShow only
    NVVE_GET_PTS_CODED_TIME,          // DShow only
    NVVE_GET_PTS_RECEIVED_TIME,       // DShow only
    NVVE_STAT_ELAPSED_TIME,           // DShow only
    NVVE_STAT_QBUF_FULLNESS,          // DShow only
    NVVE_STAT_PERF_FPS,               // DShow only
    NVVE_STAT_PERF_AVG_TIME,          // DShow only
    NVVE_DISABLE_CABAC,
    NVVE_CONFIGURE_NALU_FRAMING_TYPE,
    NVVE_DISABLE_SPS_PPS,
    NVVE_SLICE_COUNT,
    NVVE_GPU_OFFLOAD_LEVEL,
    NVVE_GPU_OFFLOAD_LEVEL_MAX,
    NVVE_MULTI_GPU,
    NVVE_GET_GPU_COUNT,
    NVVE_GET_GPU_ATTRIBUTES,
    NVVE_FORCE_GPU_SELECTION,
    NVVE_DEVICE_MEMORY_INPUT,
    NVVE_DEVICE_CTX_LOCK
};

// Rate Control Method
// Used for NVVE_RC_TYPE in SetParamValue, GetParamValue interface
// functions
enum RC_TYPE
{
    RC_CQP = 0,
    RC_VBR,
    RC_CBR,
    RC_VBR_MINQP
};

// Frame Rate
// Used for NVVE_FRAME_RATE in SetParamValue, GetParamValue interface functions
enum NVVEFrameRate
{
    NVVE_FRAME_RATE_12 = 0,
    NVVE_FRAME_RATE_12_5,
    NVVE_FRAME_RATE_14_98,
    NVVE_FRAME_RATE_15,
    NVVE_FRAME_RATE_23_97,
    NVVE_FRAME_RATE_24,
    NVVE_FRAME_RATE_25,
    NVVE_FRAME_RATE_29_97,
    NVVE_FRAME_RATE_30,
    NVVE_FRAME_RATE_50,
    NVVE_FRAME_RATE_59_94,
    NVVE_FRAME_RATE_60,
    NVVE_FRAME_RATE_NUMDEN,
    NVVE_NUM_FRAME_RATES,
    NVVE_FRAME_RATE_UNKNOWN    // Unknown/unspecified frame rate (or variable)
};

// Frame rate descriptor
// Used for NVVE_FRAME_RATE in SetParamValue, GetParamValue interface functions
typedef struct _NVVE_FrameRateDescriptor
{
    NVVE_FrameRate eFrameRate;
    int lNumerator;
    int lDenominator;
} NVVE_FrameRateDescriptor;

// Field Encoding mode
// Used for NVVE_FIELD_ENC_MODE in SetParamValue, GetParamValue interface
// functions
enum NVVE_FIELD_MODE
{
    MODE_FRAME = 0,
    MODE_FIELD_TOP_FIRST,
    MODE_FIELD_BOTTOM_FIRST,
    MODE_FIELD_PICAFF,                  // not supported
};

// Deinterlacing algorithm
// Used for NVVE_SET_DEINTERLACE in SetParamValue, GetParamValue interface
// functions
enum NVVE_DI_MODE
{
    DI_OFF,
    DI_MEDIAN,
};

// Encoding Presets
// Used for NVVE_PRESETS in SetParamValue, GetParamValue interface functions
enum NVVE_PRESETS_TARGET
{
    ENC_PRESET_PSP,
    ENC_PRESET_IPOD,
    ENC_PRESET_AVCHD,
    ENC_PRESET_BD,
    ENC_PRESET_HDV_1440,
};

// Specifies whether Display Aspect Ratio(DAR)or Sample Aspect Ratio (SAR) is
// to be used
enum NVVE_ASPECT_RATIO_TYPE                         // C-lib only
{
    ASPECT_RATIO_DAR,
    ASPECT_RATIO_SAR,
};


// Surface Formats
enum NVVE_SurfaceFormat                             // C-lib only
{
    UYVY,
    YUY2,
    YV12,
    NV12,
    IYUV
};

// Picture Structure
enum NVVE_PicStruct                                 // C-lib only
{
    TOP_FIELD = 0x1,
    BOTTOM_FIELD,
    FRAME_PICTURE
};

// Aspect Ratio Parameters
typedef struct _NVVE_AspectRatioParams              // Dshow only
{
    float fAspectRatio; // set as -1.0f for custom aspect ratio
    int iWidth;          // parameter valid only for custom aspect ratio
    int iHeight;         // parameter valid only for custom aspect ratio
    NVVE_ASPECT_RATIO_TYPE eType; // parameter valid only for custom aspect
                                  // ratio
}NVVE_AspectRatioParams;

// GPU attributes
typedef struct _NVVE_GPUAttributes
{
    int                   iGpuOrdinal;          // GPU device number
    char                  cName[256];           // string identifying GPU
                                                // device
    unsigned int          uiTotalGlobalMem;     // total global memory
                                                // available on device in bytes
    int                   iMajor;               // GPU device compute
                                                // capability major version
                                                // number
    int                   iMinor;               // GPU device compute
                                                // capability minor version
                                                // number
    int                   iClockRate;           // GPU clock frequency in
                                                // kilohertz
    int                   iMultiProcessorCount; // number of multiprocessors on
                                                // the GPU device
    NVVE_GPUOffloadLevel  MaxGpuOffloadLevel;   // max offload level supported
                                                // for this GPU device
} NVVE_GPUAttributes;

// Information passed on to EncodeFrame
typedef struct _NVEncodeFrameParams // C-lib only
{
    int Width; int Height; int Pitch;
    NVVE_SurfaceFormat SurfFmt;
    NVVE_PicStruct PictureStruc;
    BOOL topfieldfirst;
    BOOL repeatFirstField;
    BOOL progressiveFrame;
    BOOL bLast;
    unsigned char *picbuf;    // pointer to yuv buffer
};

// Information passed to OnBeginFrame
typedef struct _NVVE_BeginFrameInfo            // C-lib only
{
    int nFrameNumber;    // Frame Number
    int nPicType;        // Picture Type

// Information passed to OnEndFrame
typedef struct _NVVE_EndFrameInfo              // C-lib only
{
    int nFrameNumber;    // Frame Number
    int nPicType;        // Picture Type
}

typedef struct _CUcontextlock_st *CUvideoctxlock;

// DirectShow Filter INVVESetting interface
DECLARE_INTERFACE_(INVVESetting, IUnknown)
{
    STDMETHOD(IsSupportedCodec)(THIS_ DWORD dwCodecType) PURE;
    STDMETHOD(IsSupportedCodecProfile)(THIS_ DWORD dwCodecType,
                                       DWORD dwProfileType) PURE;
    STDMETHOD(SetCodecType)(THIS_ DWORD dwCodecType) PURE;
    STDMETHOD(GetCodecType)(THIS_ DWORD *pdwCodecType) PURE;
    STDMETHOD(IsSupportedParam)(THIS_ DWORD dwParamType) PURE;
    STDMETHOD(SetParamValue)(THIS_ DWORD dwParamType, LPVOID pData) PURE;
    STDMETHOD(GetParamValue)(THIS_ DWORD dwParamType, LPVOID pData) PURE;
    STDMETHOD(SetDefaultParam)(THIS_ void) PURE;
    STDMETHOD(GetSPSPPS)(THIS_ unsigned char *pSPSPPSbfr, int nSizeSPSPPSbfr,
                         int *pDatasize) PURE;
};

// C-library API Callback structures and functions
typedef void (_stdcall *PFNACQUIREBITSTREAM)(int nBytesInBuffer,
                                             unsigned char *cb,
                                             void *pUserdata);
typedef void (_stdcall *PFNRELEASEBITSTREAM)(int nBytesInBuffer,
                                             unsigned char *cb,
                                             void *pUserdata);
typedef void (_stdcall *PFNONBEGINFRAME)(const NVVE_BeginFrameInfo *pbfi,
                                         void *pUserdata);
typedef void (_stdcall *PFNONENDFRAME)(const NVVE_EndFrameInfo *pefi,
                                       void *pUserdata);

typedef _struct NVVE_CallbackParams
{
    PFNACQUIREBITSTREAM pfnacquirebitstream;
    PFNRELEASEBITSTREAM pfnreleasebitstream;
    PFNONBEGINFRAME pfnonbeginframe;
    PFNONENDFRAME pfnonendframe;
} NVVE_CallbackParams;

typedef void *NVEncoder;

HRESULT __stdcall NVCreateEncoder(NVEncoder *pNVEncoder);
HRESULT __stdcall NVDestroyEncoder(NVEncoder hNVEncoder);
HRESULT __stdcall NVIsSupportedCodec(NVEncoder hNVEncoder, DWORD dwCodecType);
HRESULT __stdcall NVIsSupportedCodecProfile(NVEncoder hNVEncoder,
                                            DWORD dwCodecType,
                                            DWORD dwProfileType);
HRESULT __stdcall NVSetCodec(NVEncoder hNVEncoder, DWORD dwCodecType);
HRESULT __stdcall NVGetCodec(NVEncoder hNVEncoder, DWORD *pdwCodecType);
HRESULT __stdcall NVIsSupportedParam(NVEncoder hNVEncoder, DWORD dwParamType);
HRESULT __stdcall NVSetParamValue(NVEncoder hNVEncoder, DWORD dwParamType,
                                  LPVOID pData);
HRESULT __stdcall NVGetParamValue(NVEncoder hNVEncoder, DWORD dwParamType,
                                  LPVOID pData);
HRESULT __stdcall NVSetDefaultParam(NVEncoder hNVEncoder);
HRESULT __stdcall NVCreateHWEncoder(NVEncoder hNVEncoder);
HRESULT __stdcall NVGetSPSPPS(NVEncoder hNVEncoder, unsigned char *pSPSPPSbfr,
                              int nSizeSPSPPSbfr, int *pDatasize);
HRESULT __stdcall NVEncodeFrame(NVEncoder hNVEncoder,
                                NVVE_EncodeFrameParams *pFrmIn,
                                DWORD flag, void *pData);
HRESULT __stdcall NVGetHWEncodeCaps(void);
void __stdcall NVRegisterCB(NVEncoder hNVEncoder, NVVE_CallbackParams cb,
                            void *pUserdata));
```