



**NVENC - NVIDIA Video Encoder API  
Reference Manual**

**November 10, 2015**

**Version 6.0**





# Contents

<b>1</b>	<b>Legal Notice</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Modules . . . . .	3
<b>3</b>	<b>Data Structure Index</b>	<b>5</b>
3.1	Data Structures . . . . .	5
<b>4</b>	<b>Module Documentation</b>	<b>7</b>
4.1	NvEncodeAPI Data structures . . . . .	7
4.1.1	Define Documentation . . . . .	10
4.1.1.1	NV_ENC_CAPS_PARAM_VER . . . . .	10
4.1.1.2	NV_ENC_CONFIG_VER . . . . .	10
4.1.1.3	NV_ENC_CREATE_BITSTREAM_BUFFER_VER . . . . .	10
4.1.1.4	NV_ENC_CREATE_INPUT_BUFFER_VER . . . . .	10
4.1.1.5	NV_ENC_CREATE_MV_BUFFER_VER . . . . .	10
4.1.1.6	NV_ENC_EVENT_PARAMS_VER . . . . .	11
4.1.1.7	NV_ENC_INITIALIZE_PARAMS_VER . . . . .	11
4.1.1.8	NV_ENC_LOCK_BITSTREAM_VER . . . . .	11
4.1.1.9	NV_ENC_LOCK_INPUT_BUFFER_VER . . . . .	11
4.1.1.10	NV_ENC_MAP_INPUT_RESOURCE_VER . . . . .	11
4.1.1.11	NV_ENC_MEONLY_PARAMS_VER . . . . .	11
4.1.1.12	NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER . . . . .	11
4.1.1.13	NV_ENC_PARAMS_RC_CBR2 . . . . .	11
4.1.1.14	NV_ENC_PIC_PARAMS_VER . . . . .	11
4.1.1.15	NV_ENC_PRESET_CONFIG_VER . . . . .	11
4.1.1.16	NV_ENC_RC_PARAMS_VER . . . . .	11
4.1.1.17	NV_ENC_RECONFIGURE_PARAMS_VER . . . . .	12
4.1.1.18	NV_ENC_REGISTER_RESOURCE_VER . . . . .	12

4.1.1.19	NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER	12
4.1.1.20	NV_ENC_STAT_VER	12
4.1.2	Enumeration Type Documentation	12
4.1.2.1	NV_ENC_BUFFER_FORMAT	12
4.1.2.2	NV_ENC_CAPS	12
4.1.2.3	NV_ENC_DEVICE_TYPE	15
4.1.2.4	NV_ENC_H264_ADAPTIVE_TRANSFORM_MODE	15
4.1.2.5	NV_ENC_H264_BDIRECT_MODE	15
4.1.2.6	NV_ENC_H264_ENTROPY_CODING_MODE	16
4.1.2.7	NV_ENC_H264_FMO_MODE	16
4.1.2.8	NV_ENC_HEVC_CUSIZE	16
4.1.2.9	NV_ENC_INPUT_RESOURCE_TYPE	16
4.1.2.10	NV_ENC_LEVEL	16
4.1.2.11	NV_ENC_MEMORY_HEAP	16
4.1.2.12	NV_ENC_MV_PRECISION	17
4.1.2.13	NV_ENC_PARAMS_FRAME_FIELD_MODE	17
4.1.2.14	NV_ENC_PARAMS_RC_MODE	17
4.1.2.15	NV_ENC_PIC_FLAGS	17
4.1.2.16	NV_ENC_PIC_STRUCT	18
4.1.2.17	NV_ENC_PIC_TYPE	18
4.1.2.18	NV_ENC_STEREO_PACKING_MODE	18
4.1.2.19	NVENCSTATUS	18
4.2	NvEncodeAPI Functions	21
4.2.1	Function Documentation	24
4.2.1.1	NvEncCreateBitstreamBuffer	24
4.2.1.2	NvEncCreateInputBuffer	24
4.2.1.3	NvEncCreateMVBuffer	25
4.2.1.4	NvEncDestroyBitstreamBuffer	25
4.2.1.5	NvEncDestroyEncoder	25
4.2.1.6	NvEncDestroyInputBuffer	26
4.2.1.7	NvEncDestroyMVBuffer	26
4.2.1.8	NvEncEncodePicture	27
4.2.1.9	NvEncGetEncodeCaps	30
4.2.1.10	NvEncGetEncodeGUIDCount	30
4.2.1.11	NvEncGetEncodeGUIDs	30
4.2.1.12	NvEncGetEncodePresetConfig	31
4.2.1.13	NvEncGetEncodePresetCount	31

4.2.1.14	NvEncGetEncodePresetGUIDs	32
4.2.1.15	NvEncGetEncodeProfileGUIDCount	32
4.2.1.16	NvEncGetEncodeProfileGUIDs	33
4.2.1.17	NvEncGetEncodeStats	33
4.2.1.18	NvEncGetInputFormatCount	34
4.2.1.19	NvEncGetInputFormats	34
4.2.1.20	NvEncGetSequenceParams	35
4.2.1.21	NvEncInitializeEncoder	35
4.2.1.22	NvEncInvalidateRefFrames	37
4.2.1.23	NvEncLockBitstream	37
4.2.1.24	NvEncLockInputBuffer	38
4.2.1.25	NvEncMapInputResource	39
4.2.1.26	NvEncodeAPICreateInstance	39
4.2.1.27	NvEncOpenEncodeSession	39
4.2.1.28	NvEncOpenEncodeSessionEx	40
4.2.1.29	NvEncReconfigureEncoder	40
4.2.1.30	NvEncRegisterAsyncEvent	41
4.2.1.31	NvEncRegisterResource	41
4.2.1.32	NvEncRunMotionEstimationOnly	42
4.2.1.33	NvEncUnlockBitstream	42
4.2.1.34	NvEncUnlockInputBuffer	43
4.2.1.35	NvEncUnmapInputResource	43
4.2.1.36	NvEncUnregisterAsyncEvent	44
4.2.1.37	NvEncUnregisterResource	44
<b>5</b>	<b>Data Structure Documentation</b>	<b>45</b>
5.1	_NV_ENC_CODEC_CONFIG Struct Reference	45
5.1.1	Detailed Description	45
5.2	_NV_ENC_CONFIG Struct Reference	46
5.2.1	Detailed Description	46
5.3	_NV_ENC_CONFIG_H264 Struct Reference	47
5.3.1	Detailed Description	47
5.4	_NV_ENC_CONFIG_H264_VUI_PARAMETERS Struct Reference	48
5.4.1	Detailed Description	48
5.5	_NV_ENC_CONFIG_HEVC Struct Reference	49
5.5.1	Detailed Description	49
5.6	_NV_ENC_INITIALIZE_PARAMS Struct Reference	50

5.6.1 Detailed Description . . . . .	50
5.7 <a href="#">_NV_ENC_LOCK_BITSTREAM Struct Reference</a> . . . . .	51
5.7.1 Detailed Description . . . . .	51
5.8 <a href="#">_NV_ENC_LOCK_INPUT_BUFFER Struct Reference</a> . . . . .	52
5.8.1 Detailed Description . . . . .	52
5.9 <a href="#">_NV_ENC_MAP_INPUT_RESOURCE Struct Reference</a> . . . . .	53
5.9.1 Detailed Description . . . . .	53
5.10 <a href="#">_NV_ENC_MEONLY_PARAMS Struct Reference</a> . . . . .	54
5.10.1 Detailed Description . . . . .	54
5.11 <a href="#">_NV_ENC_PIC_PARAMS Struct Reference</a> . . . . .	55
5.11.1 Detailed Description . . . . .	55
5.12 <a href="#">_NV_ENC_PIC_PARAMS_H264 Struct Reference</a> . . . . .	56
5.12.1 Detailed Description . . . . .	56
5.13 <a href="#">_NV_ENC_PIC_PARAMS_HEVC Struct Reference</a> . . . . .	57
5.13.1 Detailed Description . . . . .	57
5.14 <a href="#">_NV_ENC_PRESET_CONFIG Struct Reference</a> . . . . .	58
5.14.1 Detailed Description . . . . .	58
5.15 <a href="#">_NV_ENC_RECONFIGURE_PARAMS Struct Reference</a> . . . . .	59
5.15.1 Detailed Description . . . . .	59
5.16 <a href="#">_NV_ENC_REGISTER_RESOURCE Struct Reference</a> . . . . .	60
5.16.1 Detailed Description . . . . .	60
5.17 <a href="#">_NV_ENC_SEL_PAYLOAD Struct Reference</a> . . . . .	61
5.17.1 Detailed Description . . . . .	61
5.18 <a href="#">_NV_ENC_SEQUENCE_PARAM_PAYLOAD Struct Reference</a> . . . . .	62
5.18.1 Detailed Description . . . . .	62
5.19 <a href="#">_NV_ENC_STAT Struct Reference</a> . . . . .	63
5.19.1 Detailed Description . . . . .	63
5.20 <a href="#">_NVENC_EXTERNAL_ME_HINT Struct Reference</a> . . . . .	64
5.20.1 Detailed Description . . . . .	64
5.21 <a href="#">_NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE Struct Reference</a> . . . . .	65
5.21.1 Detailed Description . . . . .	65
5.22 <a href="#">_NVENC_RECT Struct Reference</a> . . . . .	66
5.22.1 Detailed Description . . . . .	66
5.23 <a href="#">GUID Struct Reference</a> . . . . .	67
5.23.1 Detailed Description . . . . .	67
5.23.2 <a href="#">Field Documentation</a> . . . . .	67
5.23.2.1 <a href="#">Data1</a> . . . . .	67

---

5.23.2.2	Data2	67
5.23.2.3	Data3	67
5.23.2.4	Data4	67
5.24	NV_ENC_CAPS_PARAM Struct Reference	68
5.24.1	Detailed Description	68
5.24.2	Field Documentation	68
5.24.2.1	capsToQuery	68
5.24.2.2	reserved	68
5.24.2.3	version	68
5.25	NV_ENC_CODEC_PIC_PARAMS Union Reference	69
5.25.1	Detailed Description	69
5.25.2	Field Documentation	69
5.25.2.1	h264PicParams	69
5.25.2.2	hevcPicParams	69
5.25.2.3	reserved	69
5.26	NV_ENC_CREATE_BITSTREAM_BUFFER Struct Reference	70
5.26.1	Detailed Description	70
5.26.2	Field Documentation	70
5.26.2.1	bitstreamBuffer	70
5.26.2.2	bitstreamBufferPtr	70
5.26.2.3	memoryHeap	70
5.26.2.4	reserved	70
5.26.2.5	reserved1	70
5.26.2.6	reserved2	70
5.26.2.7	size	71
5.26.2.8	version	71
5.27	NV_ENC_CREATE_INPUT_BUFFER Struct Reference	72
5.27.1	Detailed Description	72
5.27.2	Field Documentation	72
5.27.2.1	bufferFmt	72
5.27.2.2	height	72
5.27.2.3	inputBuffer	72
5.27.2.4	memoryHeap	72
5.27.2.5	pSysMemBuffer	72
5.27.2.6	reserved	72
5.27.2.7	reserved1	73
5.27.2.8	reserved2	73

5.27.2.9	version	73
5.27.2.10	width	73
5.28	NV_ENC_CREATE_MV_BUFFER Struct Reference	74
5.28.1	Detailed Description	74
5.28.2	Field Documentation	74
5.28.2.1	MVBuffer	74
5.28.2.2	reserved1	74
5.28.2.3	reserved2	74
5.28.2.4	version	74
5.29	NV_ENC_EVENT_PARAMS Struct Reference	75
5.29.1	Detailed Description	75
5.29.2	Field Documentation	75
5.29.2.1	completionEvent	75
5.29.2.2	reserved	75
5.29.2.3	reserved1	75
5.29.2.4	reserved2	75
5.29.2.5	version	75
5.30	NV_ENC_H264_MV_DATA Struct Reference	76
5.30.1	Detailed Description	76
5.30.2	Field Documentation	76
5.30.2.1	mb_type	76
5.30.2.2	MV	76
5.30.2.3	partitionType	76
5.30.2.4	reserved	76
5.31	NV_ENC_MVECTOR Struct Reference	77
5.31.1	Detailed Description	77
5.31.2	Field Documentation	77
5.31.2.1	mvx	77
5.31.2.2	mvy	77
5.32	NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS Struct Reference	78
5.32.1	Detailed Description	78
5.32.2	Field Documentation	78
5.32.2.1	apiVersion	78
5.32.2.2	device	78
5.32.2.3	deviceType	78
5.32.2.4	reserved	78
5.32.2.5	reserved1	78



5.32.2.6 reserved2 . . . . .	78
5.32.2.7 version . . . . .	79
5.33 NV_ENC_QP Struct Reference . . . . .	80
5.33.1 Detailed Description . . . . .	80
5.34 NV_ENC_RC_PARAMS Struct Reference . . . . .	81
5.34.1 Detailed Description . . . . .	81
5.34.2 Field Documentation . . . . .	81
5.34.2.1 averageBitRate . . . . .	81
5.34.2.2 constQP . . . . .	81
5.34.2.3 enableAQ . . . . .	81
5.34.2.4 enableExtQPDeltaMap . . . . .	81
5.34.2.5 enableInitialRCQP . . . . .	82
5.34.2.6 enableMaxQP . . . . .	82
5.34.2.7 enableMinQP . . . . .	82
5.34.2.8 initialRCQP . . . . .	82
5.34.2.9 maxBitRate . . . . .	82
5.34.2.10 maxQP . . . . .	82
5.34.2.11 minQP . . . . .	82
5.34.2.12 rateControlMode . . . . .	82
5.34.2.13 reservedBitFields . . . . .	82
5.34.2.14 temporallayerIdxMask . . . . .	82
5.34.2.15 temporalLayerQP . . . . .	82
5.34.2.16 vbvBufferSize . . . . .	83
5.34.2.17 vbvInitialDelay . . . . .	83
5.35 NV_ENCODE_API_FUNCTION_LIST Struct Reference . . . . .	84
5.35.1 Detailed Description . . . . .	84
5.35.2 Field Documentation . . . . .	85
5.35.2.1 nvEncCreateBitstreamBuffer . . . . .	85
5.35.2.2 nvEncCreateInputBuffer . . . . .	85
5.35.2.3 nvEncCreateMVBuffer . . . . .	85
5.35.2.4 nvEncDestroyBitstreamBuffer . . . . .	85
5.35.2.5 nvEncDestroyEncoder . . . . .	85
5.35.2.6 nvEncDestroyInputBuffer . . . . .	85
5.35.2.7 nvEncDestroyMVBuffer . . . . .	85
5.35.2.8 nvEncEncodePicture . . . . .	85
5.35.2.9 nvEncGetEncodeCaps . . . . .	85
5.35.2.10 nvEncGetEncodeGUIDCount . . . . .	85

---

5.35.2.11 nvEncGetEncodeGUIDs . . . . .	86
5.35.2.12 nvEncGetEncodePresetConfig . . . . .	86
5.35.2.13 nvEncGetEncodePresetCount . . . . .	86
5.35.2.14 nvEncGetEncodePresetGUIDs . . . . .	86
5.35.2.15 nvEncGetEncodeProfileGUIDCount . . . . .	86
5.35.2.16 nvEncGetEncodeProfileGUIDs . . . . .	86
5.35.2.17 nvEncGetEncodeStats . . . . .	86
5.35.2.18 nvEncGetInputFormatCount . . . . .	86
5.35.2.19 nvEncGetInputFormats . . . . .	86
5.35.2.20 nvEncGetSequenceParams . . . . .	86
5.35.2.21 nvEncInitializeEncoder . . . . .	87
5.35.2.22 nvEncInvalidateRefFrames . . . . .	87
5.35.2.23 nvEncLockBitstream . . . . .	87
5.35.2.24 nvEncLockInputBuffer . . . . .	87
5.35.2.25 nvEncMapInputResource . . . . .	87
5.35.2.26 nvEncOpenEncodeSession . . . . .	87
5.35.2.27 nvEncOpenEncodeSessionEx . . . . .	87
5.35.2.28 nvEncReconfigureEncoder . . . . .	87
5.35.2.29 nvEncRegisterAsyncEvent . . . . .	87
5.35.2.30 nvEncRegisterResource . . . . .	87
5.35.2.31 nvEncRunMotionEstimationOnly . . . . .	88
5.35.2.32 nvEncUnlockBitstream . . . . .	88
5.35.2.33 nvEncUnlockInputBuffer . . . . .	88
5.35.2.34 nvEncUnmapInputResource . . . . .	88
5.35.2.35 nvEncUnregisterAsyncEvent . . . . .	88
5.35.2.36 nvEncUnregisterResource . . . . .	88
5.35.2.37 reserved . . . . .	88
5.35.2.38 reserved2 . . . . .	88
5.35.2.39 version . . . . .	88

# Chapter 1

## Legal Notice

**Copyright** (c) 2011-2014 NVIDIA Corporation. All rights reserved.

### **Notice**

This source code and/or documentation ("Licensed Deliverables") are subject to NVIDIA intellectual property rights under U.S. and international Copyright laws.

These Licensed Deliverables contained herein is PROPRIETARY and to NVIDIA and is being provided under the terms and conditions of a form of NVIDIA software license agreement by and between NVIDIA and Licensee ("License Agreement") or electronically accepted by Licensee. Notwithstanding any terms or conditions to the contrary in the License Agreement, reproduction or disclosure of the Licensed Deliverables to any third party without the express written consent of NVIDIA is prohibited.

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND. NVIDIA DISCLAIMS ALL WARRANTIES WITH REGARD TO THESE LICENSED DELIVERABLES, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE. NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY IN THE LICENSE AGREEMENT, IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THESE LICENSED DELIVERABLES.

Information furnished is believed to be accurate and reliable. However, NVIDIA assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties, which may result from its use. No License is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in the software are subject to change without notice. This publication supersedes and replaces all other information previously supplied.

NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

U.S. Government End Users. These Licensed Deliverables are a "commercial item" as that term is defined at 48 C.F.R. 2.101 (OCT \* 1995), consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (SEPT 1995) and is provided to the U.S. Government only as a commercial end item. Consistent with 48 C.F.R.12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire the Licensed Deliverables with only those rights set forth herein.

Any use of the Licensed Deliverables in individual and commercial software must include, in the user documentation and internal comments to the code, the above Disclaimer and U.S. Government End Users Notice.

**Trademarks**

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Microsoft, Windows, and the Windows logo are registered trademarks of Microsoft Corporation.

Other company and product names may be trademarks or registered trademarks of the respective companies with which they are associated.

# Chapter 2

## Module Index

### 2.1 Modules

Here is a list of all modules:

NvEncodeAPI Data structures . . . . .	7
NvEncodeAPI Functions . . . . .	21



# Chapter 3

## Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_NV_ENC_CODEC_CONFIG</a>	45
<a href="#">_NV_ENC_CONFIG</a>	46
<a href="#">_NV_ENC_CONFIG_H264</a>	47
<a href="#">_NV_ENC_CONFIG_H264_VUI_PARAMETERS</a>	48
<a href="#">_NV_ENC_CONFIG_HEVC</a>	49
<a href="#">_NV_ENC_INITIALIZE_PARAMS</a>	50
<a href="#">_NV_ENC_LOCK_BITSTREAM</a>	51
<a href="#">_NV_ENC_LOCK_INPUT_BUFFER</a>	52
<a href="#">_NV_ENC_MAP_INPUT_RESOURCE</a>	53
<a href="#">_NV_ENC_MEONLY_PARAMS</a>	54
<a href="#">_NV_ENC_PIC_PARAMS</a>	55
<a href="#">_NV_ENC_PIC_PARAMS_H264</a>	56
<a href="#">_NV_ENC_PIC_PARAMS_HEVC</a>	57
<a href="#">_NV_ENC_PRESET_CONFIG</a>	58
<a href="#">_NV_ENC_RECONFIGURE_PARAMS</a>	59
<a href="#">_NV_ENC_REGISTER_RESOURCE</a>	60
<a href="#">_NV_ENC_SEI_PAYLOAD</a>	61
<a href="#">_NV_ENC_SEQUENCE_PARAM_PAYLOAD</a>	62
<a href="#">_NV_ENC_STAT</a>	63
<a href="#">_NVENC_EXTERNAL_ME_HINT</a>	64
<a href="#">_NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE</a>	65
<a href="#">_NVENC_RECT</a>	66
<a href="#">GUID</a>	67
<a href="#">NV_ENC_CAPS_PARAM</a>	68
<a href="#">NV_ENC_CODEC_PIC_PARAMS</a>	69
<a href="#">NV_ENC_CREATE_BITSTREAM_BUFFER</a>	70
<a href="#">NV_ENC_CREATE_INPUT_BUFFER</a>	72
<a href="#">NV_ENC_CREATE_MV_BUFFER</a>	74
<a href="#">NV_ENC_EVENT_PARAMS</a>	75
<a href="#">NV_ENC_H264_MV_DATA</a>	76
<a href="#">NV_ENC_MVECTOR</a>	77
<a href="#">NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS</a>	78
<a href="#">NV_ENC_QP</a>	80

NV_ENC_RC_PARAMS .....	81
NV_ENCODE_API_FUNCTION_LIST .....	84



# Chapter 4

## Module Documentation

### 4.1 NvEncodeAPI Data structures

#### Data Structures

- struct [GUID](#)
- struct [NV\\_ENC\\_CAPS\\_PARAM](#)
- struct [NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER](#)
- struct [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER](#)
- struct [NV\\_ENC\\_MVECTOR](#)
- struct [NV\\_ENC\\_H264\\_MV\\_DATA](#)
- struct [NV\\_ENC\\_CREATE\\_MV\\_BUFFER](#)
- struct [NV\\_ENC\\_QP](#)
- struct [NV\\_ENC\\_RC\\_PARAMS](#)
- union [NV\\_ENC\\_CODEC\\_PIC\\_PARAMS](#)
- struct [NV\\_ENC\\_EVENT\\_PARAMS](#)
- struct [NV\\_ENC\\_OPEN\\_ENCODE\\_SESSION\\_EX\\_PARAMS](#)
- struct [NV\\_ENCODE\\_API\\_FUNCTION\\_LIST](#)
- struct [\\_NVENC\\_RECT](#)
- struct [\\_NV\\_ENC\\_CONFIG\\_H264\\_VUI\\_PARAMETERS](#)
- struct [\\_NVENC\\_EXTERNAL\\_ME\\_HINT\\_COUNTS\\_PER\\_BLOCKTYPE](#)
- struct [\\_NVENC\\_EXTERNAL\\_ME\\_HINT](#)
- struct [\\_NV\\_ENC\\_CONFIG\\_H264](#)
- struct [\\_NV\\_ENC\\_CONFIG\\_HEVC](#)
- struct [\\_NV\\_ENC\\_CODEC\\_CONFIG](#)
- struct [\\_NV\\_ENC\\_CONFIG](#)
- struct [\\_NV\\_ENC\\_INITIALIZE\\_PARAMS](#)
- struct [\\_NV\\_ENC\\_RECONFIGURE\\_PARAMS](#)
- struct [\\_NV\\_ENC\\_PRESET\\_CONFIG](#)
- struct [\\_NV\\_ENC\\_SEI\\_PAYLOAD](#)
- struct [\\_NV\\_ENC\\_PIC\\_PARAMS\\_H264](#)
- struct [\\_NV\\_ENC\\_PIC\\_PARAMS\\_HEVC](#)
- struct [\\_NV\\_ENC\\_PIC\\_PARAMS](#)
- struct [\\_NV\\_ENC\\_MEONLY\\_PARAMS](#)
- struct [\\_NV\\_ENC\\_LOCK\\_BITSTREAM](#)
- struct [\\_NV\\_ENC\\_LOCK\\_INPUT\\_BUFFER](#)

- struct `_NV_ENC_MAP_INPUT_RESOURCE`
- struct `_NV_ENC_REGISTER_RESOURCE`
- struct `_NV_ENC_STAT`
- struct `_NV_ENC_SEQUENCE_PARAM_PAYLOAD`

## Defines

- #define `NV_ENC_PARAMS_RC_CBR2` `NV_ENC_PARAMS_RC_CBR`
- #define `NV_ENC_CAPS_PARAM_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_CREATE_INPUT_BUFFER_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_CREATE_BITSTREAM_BUFFER_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_CREATE_MV_BUFFER_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_RC_PARAMS_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_CONFIG_VER` `(NVENCAPI_STRUCT_VERSION(6) | ( 1<<31 ))`
- #define `NV_ENC_INITIALIZE_PARAMS_VER` `(NVENCAPI_STRUCT_VERSION(5) | ( 1<<31 ))`
- #define `NV_ENC_RECONFIGURE_PARAMS_VER` `(NVENCAPI_STRUCT_VERSION(1) | ( 1<<31 ))`
- #define `NV_ENC_PRESET_CONFIG_VER` `(NVENCAPI_STRUCT_VERSION(4) | ( 1<<31 ))`
- #define `NV_ENC_PIC_PARAMS_VER` `(NVENCAPI_STRUCT_VERSION(4) | ( 1<<31 ))`
- #define `NV_ENC_MEONLY_PARAMS_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_LOCK_BITSTREAM_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_LOCK_INPUT_BUFFER_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_MAP_INPUT_RESOURCE_VER` `NVENCAPI_STRUCT_VERSION(4)`
- #define `NV_ENC_REGISTER_RESOURCE_VER` `NVENCAPI_STRUCT_VERSION(3)`
- #define `NV_ENC_STAT_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_EVENT_PARAMS_VER` `NVENCAPI_STRUCT_VERSION(1)`
- #define `NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER` `NVENCAPI_STRUCT_VERSION(1)`

## Enumerations

- enum `NV_ENC_PARAMS_FRAME_FIELD_MODE` { `NV_ENC_PARAMS_FRAME_FIELD_MODE_FRAME` = 0x01, `NV_ENC_PARAMS_FRAME_FIELD_MODE_FIELD` = 0x02, `NV_ENC_PARAMS_FRAME_FIELD_MODE_MBAFF` = 0x03 }
- enum `NV_ENC_PARAMS_RC_MODE` {  
`NV_ENC_PARAMS_RC_CONSTQP` = 0x0, `NV_ENC_PARAMS_RC_VBR` = 0x1, `NV_ENC_PARAMS_RC_CBR` = 0x2, `NV_ENC_PARAMS_RC_VBR_MINQP` = 0x4,  
`NV_ENC_PARAMS_RC_2_PASS_QUALITY` = 0x8, `NV_ENC_PARAMS_RC_2_PASS_FRAMESIZE_CAP` = 0x10, `NV_ENC_PARAMS_RC_2_PASS_VBR` = 0x20 }
- enum `NV_ENC_PIC_STRUCT` { `NV_ENC_PIC_STRUCT_FRAME` = 0x01, `NV_ENC_PIC_STRUCT_FIELD_TOP_BOTTOM` = 0x02, `NV_ENC_PIC_STRUCT_FIELD_BOTTOM_TOP` = 0x03 }
- enum `NV_ENC_PIC_TYPE` {  
`NV_ENC_PIC_TYPE_P` = 0x0, `NV_ENC_PIC_TYPE_B` = 0x01, `NV_ENC_PIC_TYPE_I` = 0x02, `NV_ENC_PIC_TYPE_IDR` = 0x03,  
`NV_ENC_PIC_TYPE_BI` = 0x04, `NV_ENC_PIC_TYPE_SKIPPED` = 0x05, `NV_ENC_PIC_TYPE_INTRA_REFRESH` = 0x06, `NV_ENC_PIC_TYPE_UNKNOWN` = 0xFF }
- enum `NV_ENC_MV_PRECISION` { `NV_ENC_MV_PRECISION_DEFAULT` = 0x0, `NV_ENC_MV_PRECISION_FULL_PEL` = 0x01, `NV_ENC_MV_PRECISION_HALF_PEL` = 0x02, `NV_ENC_MV_PRECISION_QUARTER_PEL` = 0x03 }

- enum NV\_ENC\_BUFFER\_FORMAT {
  - NV\_ENC\_BUFFER\_FORMAT\_UNDEFINED = 0x0, NV\_ENC\_BUFFER\_FORMAT\_NV12 = 0x1, NV\_ENC\_BUFFER\_FORMAT\_YV12 = 0x10, NV\_ENC\_BUFFER\_FORMAT\_IYUV = 0x100,
  - NV\_ENC\_BUFFER\_FORMAT\_YUV444 = 0x1000, NV\_ENC\_BUFFER\_FORMAT\_ARGB = 0x1000000, NV\_ENC\_BUFFER\_FORMAT\_ARGB10 = 0x2000000, NV\_ENC\_BUFFER\_FORMAT\_AYUV = 0x4000000
- enum NV\_ENC\_LEVEL
- enum NVENCSTATUS {
  - NV\_ENC\_SUCCESS, NV\_ENC\_ERR\_NO\_ENCODE\_DEVICE, NV\_ENC\_ERR\_UNSUPPORTED\_DEVICE, NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE,
  - NV\_ENC\_ERR\_INVALID\_DEVICE, NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST, NV\_ENC\_ERR\_INVALID\_PTR, NV\_ENC\_ERR\_INVALID\_EVENT,
  - NV\_ENC\_ERR\_INVALID\_PARAM, NV\_ENC\_ERR\_INVALID\_CALL, NV\_ENC\_ERR\_OUT\_OF\_MEMORY, NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED,
  - NV\_ENC\_ERR\_UNSUPPORTED\_PARAM, NV\_ENC\_ERR\_LOCK\_BUSY, NV\_ENC\_ERR\_NOT\_ENOUGH\_BUFFER, NV\_ENC\_ERR\_INVALID\_VERSION,
  - NV\_ENC\_ERR\_MAP\_FAILED, NV\_ENC\_ERR\_NEED\_MORE\_INPUT, NV\_ENC\_ERR\_ENCODER\_BUSY, NV\_ENC\_ERR\_EVENT\_NOT\_REGISTERD,
  - NV\_ENC\_ERR\_GENERIC, NV\_ENC\_ERR\_INCOMPATIBLE\_CLIENT\_KEY, NV\_ENC\_ERR\_UNIMPLEMENTED, NV\_ENC\_ERR\_RESOURCE\_REGISTER\_FAILED,
  - NV\_ENC\_ERR\_RESOURCE\_NOT\_REGISTERED, NV\_ENC\_ERR\_RESOURCE\_NOT\_MAPPED }
- enum NV\_ENC\_PIC\_FLAGS { NV\_ENC\_PIC\_FLAG\_FORCEINTRA = 0x1, NV\_ENC\_PIC\_FLAG\_FORCEIDR = 0x2, NV\_ENC\_PIC\_FLAG\_OUTPUT\_SPSPPS = 0x4, NV\_ENC\_PIC\_FLAG\_EOS = 0x8 }
- enum NV\_ENC\_MEMORY\_HEAP { NV\_ENC\_MEMORY\_HEAP\_AUTOSELECT = 0, NV\_ENC\_MEMORY\_HEAP\_VID = 1, NV\_ENC\_MEMORY\_HEAP\_SYSTEMEM\_CACHED = 2, NV\_ENC\_MEMORY\_HEAP\_SYSTEMEM\_UNCACHED = 3 }
- enum NV\_ENC\_H264\_ENTROPY\_CODING\_MODE { NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_AUTOSELECT = 0x0, NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_CABAC = 0x1, NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_CAVLC = 0x2 }
- enum NV\_ENC\_H264\_BDIRECT\_MODE { NV\_ENC\_H264\_BDIRECT\_MODE\_AUTOSELECT = 0x0, NV\_ENC\_H264\_BDIRECT\_MODE\_DISABLE = 0x1, NV\_ENC\_H264\_BDIRECT\_MODE\_TEMPORAL = 0x2, NV\_ENC\_H264\_BDIRECT\_MODE\_SPATIAL = 0x3 }
- enum NV\_ENC\_H264\_FMO\_MODE { NV\_ENC\_H264\_FMO\_AUTOSELECT = 0x0, NV\_ENC\_H264\_FMO\_ENABLE = 0x1, NV\_ENC\_H264\_FMO\_DISABLE = 0x2 }
- enum NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_MODE { NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_AUTOSELECT = 0x0, NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_DISABLE = 0x1, NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_ENABLE = 0x2 }
- enum NV\_ENC\_STEREO\_PACKING\_MODE {
  - NV\_ENC\_STEREO\_PACKING\_MODE\_NONE = 0x0, NV\_ENC\_STEREO\_PACKING\_MODE\_CHECKERBOARD = 0x1, NV\_ENC\_STEREO\_PACKING\_MODE\_COLINTERLEAVE = 0x2, NV\_ENC\_STEREO\_PACKING\_MODE\_ROWINTERLEAVE = 0x3,
  - NV\_ENC\_STEREO\_PACKING\_MODE\_SIDE\_BY\_SIDE = 0x4, NV\_ENC\_STEREO\_PACKING\_MODE\_TOP\_BOTTOM = 0x5, NV\_ENC\_STEREO\_PACKING\_MODE\_FRAMESEQ = 0x6 }
- enum NV\_ENC\_INPUT\_RESOURCE\_TYPE { NV\_ENC\_INPUT\_RESOURCE\_TYPE\_DIRECTX = 0x0, NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDADEVICEPTR = 0x1, NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDAARRAY = 0x2 }
- enum NV\_ENC\_DEVICE\_TYPE { NV\_ENC\_DEVICE\_TYPE\_DIRECTX = 0x0, NV\_ENC\_DEVICE\_TYPE\_CUDA = 0x1 }

- enum `NV_ENC_CAPS` {
  - `NV_ENC_CAPS_NUM_MAX_BFRAMES`, `NV_ENC_CAPS_SUPPORTED_RATECONTROL_MODES`,
  - `NV_ENC_CAPS_SUPPORT_FIELD_ENCODING`, `NV_ENC_CAPS_SUPPORT_MONOCHROME`,
  - `NV_ENC_CAPS_SUPPORT_FMO`, `NV_ENC_CAPS_SUPPORT_QPELMV`, `NV_ENC_CAPS_SUPPORT_BDIRECT_MODE`, `NV_ENC_CAPS_SUPPORT_CABAC`,
  - `NV_ENC_CAPS_SUPPORT_ADAPTIVE_TRANSFORM`, `NV_ENC_CAPS_SUPPORT_RESERVED`,
  - `NV_ENC_CAPS_NUM_MAX_TEMPORAL_LAYERS`, `NV_ENC_CAPS_SUPPORT_HIERARCHICAL_PFRAMES`,
  - `NV_ENC_CAPS_SUPPORT_HIERARCHICAL_BFRAMES`, `NV_ENC_CAPS_LEVEL_MAX`, `NV_ENC_CAPS_LEVEL_MIN`, `NV_ENC_CAPS_SEPARATE_COLOUR_PLANE`,
  - `NV_ENC_CAPS_WIDTH_MAX`, `NV_ENC_CAPS_HEIGHT_MAX`, `NV_ENC_CAPS_SUPPORT_TEMPORAL_SVC`, `NV_ENC_CAPS_SUPPORT_DYN_RES_CHANGE`,
  - `NV_ENC_CAPS_SUPPORT_DYN_BITRATE_CHANGE`, `NV_ENC_CAPS_SUPPORT_DYN_FORCE_CONSTQP`, `NV_ENC_CAPS_SUPPORT_DYN_RCMODE_CHANGE`, `NV_ENC_CAPS_SUPPORT_SUBFRAME_READBACK`,
  - `NV_ENC_CAPS_SUPPORT_CONSTRAINED_ENCODING`, `NV_ENC_CAPS_SUPPORT_INTRA_REFRESH`, `NV_ENC_CAPS_SUPPORT_CUSTOM_VBV_BUF_SIZE`, `NV_ENC_CAPS_SUPPORT_DYNAMIC_SLICE_MODE`,
  - `NV_ENC_CAPS_SUPPORT_REF_PIC_INVALIDATION`, `NV_ENC_CAPS_PREPROC_SUPPORT`, `NV_ENC_CAPS_ASYNC_ENCODE_SUPPORT`, `NV_ENC_CAPS_MB_NUM_MAX`,
  - `NV_ENC_CAPS_MB_PER_SEC_MAX`, `NV_ENC_CAPS_SUPPORT_YUV444_ENCODE`, `NV_ENC_CAPS_SUPPORT_LOSSLESS_ENCODE`, `NV_ENC_CAPS_SUPPORT_SAO`,
  - `NV_ENC_CAPS_SUPPORT_MEONLY_MODE`, `NV_ENC_CAPS_EXPOSED_COUNT` }
- enum `NV_ENC_HEVC_CUSIZE`

### 4.1.1 Define Documentation

#### 4.1.1.1 `#define NV_ENC_CAPS_PARAM_VER NVENC_API_STRUCT_VERSION(1)`

`NV_ENC_CAPS_PARAM` struct version.

#### 4.1.1.2 `#define NV_ENC_CONFIG_VER (NVENC_API_STRUCT_VERSION(6) | (1 << 31))`

macro for constructing the version field of `_NV_ENC_CONFIG`

#### 4.1.1.3 `#define NV_ENC_CREATE_BITSTREAM_BUFFER_VER NVENC_API_STRUCT_VERSION(1)`

`NV_ENC_CREATE_BITSTREAM_BUFFER` struct version.

#### 4.1.1.4 `#define NV_ENC_CREATE_INPUT_BUFFER_VER NVENC_API_STRUCT_VERSION(1)`

`NV_ENC_CREATE_INPUT_BUFFER` struct version.

#### 4.1.1.5 `#define NV_ENC_CREATE_MV_BUFFER_VER NVENC_API_STRUCT_VERSION(1)`

`NV_ENC_CREATE_MV_BUFFER` struct version

**4.1.1.6 #define NV\_ENC\_EVENT\_PARAMS\_VER NVENC\_API\_STRUCT\_VERSION(1)**

Macro for constructing the version field of `_NV_ENC_EVENT_PARAMS`

**4.1.1.7 #define NV\_ENC\_INITIALIZE\_PARAMS\_VER (NVENC\_API\_STRUCT\_VERSION(5) | (1 << 31))**

macro for constructing the version field of `_NV_ENC_INITIALIZE_PARAMS`

**4.1.1.8 #define NV\_ENC\_LOCK\_BITSTREAM\_VER NVENC\_API\_STRUCT\_VERSION(1)**

Macro for constructing the version field of `_NV_ENC_LOCK_BITSTREAM`

**4.1.1.9 #define NV\_ENC\_LOCK\_INPUT\_BUFFER\_VER NVENC\_API\_STRUCT\_VERSION(1)**

Macro for constructing the version field of `_NV_ENC_LOCK_INPUT_BUFFER`

**4.1.1.10 #define NV\_ENC\_MAP\_INPUT\_RESOURCE\_VER NVENC\_API\_STRUCT\_VERSION(4)**

Macro for constructing the version field of `_NV_ENC_MAP_INPUT_RESOURCE`

**4.1.1.11 #define NV\_ENC\_MEONLY\_PARAMS\_VER NVENC\_API\_STRUCT\_VERSION(1)**

`NV_ENC_MEONLY_PARAMS` struct version

**4.1.1.12 #define NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS\_VER NVENC\_API\_STRUCT\_VERSION(1)**

Macro for constructing the version field of `_NV_ENC_OPEN_ENCODE_SESSIONEX_PARAMS`

**4.1.1.13 #define NV\_ENC\_PARAMS\_RC\_CBR2 NV\_ENC\_PARAMS\_RC\_CBR**

Deprecated

**4.1.1.14 #define NV\_ENC\_PIC\_PARAMS\_VER (NVENC\_API\_STRUCT\_VERSION(4) | (1 << 31))**

Macro for constructing the version field of `_NV_ENC_PIC_PARAMS`

**4.1.1.15 #define NV\_ENC\_PRESET\_CONFIG\_VER (NVENC\_API\_STRUCT\_VERSION(4) | (1 << 31))**

macro for constructing the version field of `_NV_ENC_PRESET_CONFIG`

**4.1.1.16 #define NV\_ENC\_RC\_PARAMS\_VER NVENC\_API\_STRUCT\_VERSION(1)**

macro for constructing the version field of `_NV_ENC_RC_PARAMS`

**4.1.1.17** `#define NV_ENC_RECONFIGURE_PARAMS_VER (NVENC_API_STRUCT_VERSION(1) | (1 << 31))`

macro for constructing the version field of `_NV_ENC_RECONFIGURE_PARAMS`

**4.1.1.18** `#define NV_ENC_REGISTER_RESOURCE_VER NVENC_API_STRUCT_VERSION(3)`

Macro for constructing the version field of `_NV_ENC_REGISTER_RESOURCE`

**4.1.1.19** `#define NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER NVENC_API_STRUCT_VERSION(1)`

Macro for constructing the version field of `_NV_ENC_SEQUENCE_PARAM_PAYLOAD`

**4.1.1.20** `#define NV_ENC_STAT_VER NVENC_API_STRUCT_VERSION(1)`

Macro for constructing the version field of `_NV_ENC_STAT`

## 4.1.2 Enumeration Type Documentation

### 4.1.2.1 `enum NV_ENC_BUFFER_FORMAT`

Input buffer formats

#### Enumerator:

`NV_ENC_BUFFER_FORMAT_UNDEFINED` Undefined buffer format.  
`NV_ENC_BUFFER_FORMAT_NV12` Semi-Planar YUV [UV interleaved].  
`NV_ENC_BUFFER_FORMAT_YV12` Planar YUV [YUV separate planes].  
`NV_ENC_BUFFER_FORMAT_IYUV` Packed YUV [YUV separate bytes per pixel].  
`NV_ENC_BUFFER_FORMAT_YUV444` Planar YUV [YUV separate bytes per pixel].  
`NV_ENC_BUFFER_FORMAT_ARGB` 8 bit A8R8G8B8.  
`NV_ENC_BUFFER_FORMAT_ARGB10` 10 bit packed A2R10G10B10.  
`NV_ENC_BUFFER_FORMAT_AYUV` 8 bit A8Y8U8V8.

### 4.1.2.2 `enum NV_ENC_CAPS`

Encoder capabilities enumeration.

#### Enumerator:

`NV_ENC_CAPS_NUM_MAX_BFRAMES` Maximum number of B-Frames supported.  
`NV_ENC_CAPS_SUPPORTED_RATECONTROL_MODES` Rate control modes supported.  
 The API return value is a bitmask of the values in `NV_ENC_PARAMS_RC_MODE`.  
`NV_ENC_CAPS_SUPPORT_FIELD_ENCODING` Indicates HW support for field mode encoding.  
 0 : Interlaced mode encoding is not supported.  
 1 : Interlaced field mode encoding is supported.  
 2 : Interlaced frame encoding and field mode encoding are both supported.

- NV\_ENC\_CAPS\_SUPPORT\_MONOCHROME*** Indicates HW support for monochrome mode encoding.  
0 : Monochrome mode not supported.  
1 : Monochrome mode supported.
- NV\_ENC\_CAPS\_SUPPORT\_FMO*** Indicates HW support for FMO.  
0 : FMO not supported.  
1 : FMO supported.
- NV\_ENC\_CAPS\_SUPPORT\_QPELMV*** Indicates HW capability for Quarter pel motion estimation.  
0 : QuarterPel Motion Estimation not supported.  
1 : QuarterPel Motion Estimation supported.
- NV\_ENC\_CAPS\_SUPPORT\_BDIRECT\_MODE*** H.264 specific. Indicates HW support for BDirect modes.  
0 : BDirect mode encoding not supported.  
1 : BDirect mode encoding supported.
- NV\_ENC\_CAPS\_SUPPORT\_CABAC*** H264 specific. Indicates HW support for CABAC entropy coding mode.  
0 : CABAC entropy coding not supported.  
1 : CABAC entropy coding supported.
- NV\_ENC\_CAPS\_SUPPORT\_ADAPTIVE\_TRANSFORM*** Indicates HW support for Adaptive Transform.  
0 : Adaptive Transform not supported.  
1 : Adaptive Transform supported.
- NV\_ENC\_CAPS\_SUPPORT\_RESERVED*** Reserved enum field.
- NV\_ENC\_CAPS\_NUM\_MAX\_TEMPORAL\_LAYERS*** Indicates HW support for encoding Temporal layers.  
0 : Encoding Temporal layers not supported.  
1 : Encoding Temporal layers supported.
- NV\_ENC\_CAPS\_SUPPORT\_HIERARCHICAL\_PFRAMES*** Indicates HW support for Hierarchical P frames.  
0 : Hierarchical P frames not supported.  
1 : Hierarchical P frames supported.
- NV\_ENC\_CAPS\_SUPPORT\_HIERARCHICAL\_BFRAMES*** Indicates HW support for Hierarchical B frames.  
0 : Hierarchical B frames not supported.  
1 : Hierarchical B frames supported.
- NV\_ENC\_CAPS\_LEVEL\_MAX*** Maximum Encoding level supported (See [NV\\_ENC\\_LEVEL](#) for details).
- NV\_ENC\_CAPS\_LEVEL\_MIN*** Minimum Encoding level supported (See [NV\\_ENC\\_LEVEL](#) for details).
- NV\_ENC\_CAPS\_SEPARATE\_COLOUR\_PLANE*** Indicates HW support for separate colour plane encoding.  
0 : Separate colour plane encoding not supported.  
1 : Separate colour plane encoding supported.
- NV\_ENC\_CAPS\_WIDTH\_MAX*** Maximum output width supported.
- NV\_ENC\_CAPS\_HEIGHT\_MAX*** Maximum output height supported.
- NV\_ENC\_CAPS\_SUPPORT\_TEMPORAL\_SVC*** Indicates Temporal Scalability Support.  
0 : Temporal SVC encoding not supported.  
1 : Temporal SVC encoding supported.
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_RES\_CHANGE*** Indicates Dynamic Encode Resolution Change Support.  
Support added from NvEncodeAPI version 2.0.  
0 : Dynamic Encode Resolution Change not supported.  
1 : Dynamic Encode Resolution Change supported.

- NV\_ENC\_CAPS\_SUPPORT\_DYN\_BITRATE\_CHANGE*** Indicates Dynamic Encode Bitrate Change Support. Support added from NvEncodeAPI version 2.0.  
0 : Dynamic Encode bitrate change not supported.  
1 : Dynamic Encode bitrate change supported.
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_FORCE\_CONSTQP*** Indicates Forcing Constant QP On The Fly Support. Support added from NvEncodeAPI version 2.0.  
0 : Forcing constant QP on the fly not supported.  
1 : Forcing constant QP on the fly supported.
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_RCMODE\_CHANGE*** Indicates Dynamic rate control mode Change Support.  
0 : Dynamic rate control mode change not supported.  
1 : Dynamic rate control mode change supported.
- NV\_ENC\_CAPS\_SUPPORT\_SUBFRAME\_READBACK*** Indicates Subframe readback support for slice-based encoding.  
0 : Subframe readback not supported.  
1 : Subframe readback supported.
- NV\_ENC\_CAPS\_SUPPORT\_CONSTRAINED\_ENCODING*** Indicates Constrained Encoding mode support. Support added from NvEncodeAPI version 2.0.  
0 : Constrained encoding mode not supported.  
1 : Constrained encoding mode supported. If this mode is supported client can enable this during initialization. Client can then force a picture to be coded as constrained picture where each slice in a constrained picture will have `constrained_intra_pred_flag` set to 1 and `disable_deblocking_filter_idc` will be set to 2 and prediction vectors for inter macroblocks in each slice will be restricted to the slice region.
- NV\_ENC\_CAPS\_SUPPORT\_INTRA\_REFRESH*** Indicates Intra Refresh Mode Support. Support added from NvEncodeAPI version 2.0.  
0 : Intra Refresh Mode not supported.  
1 : Intra Refresh Mode supported.
- NV\_ENC\_CAPS\_SUPPORT\_CUSTOM\_VBV\_BUF\_SIZE*** Indicates Custom VBV Buffer Size support. It can be used for capping frame size. Support added from NvEncodeAPI version 2.0.  
0 : Custom VBV buffer size specification from client, not supported.  
1 : Custom VBV buffer size specification from client, supported.
- NV\_ENC\_CAPS\_SUPPORT\_DYNAMIC\_SLICE\_MODE*** Indicates Dynamic Slice Mode Support. Support added from NvEncodeAPI version 2.0.  
0 : Dynamic Slice Mode not supported.  
1 : Dynamic Slice Mode supported.
- NV\_ENC\_CAPS\_SUPPORT\_REF\_PIC\_INVALIDATION*** Indicates Reference Picture Invalidation Support. Support added from NvEncodeAPI version 2.0.  
0 : Reference Picture Invalidation not supported.  
1 : Reference Picture Invalidation supported.
- NV\_ENC\_CAPS\_PREPROC\_SUPPORT*** Indicates support for PreProcessing. The API return value is a bit-mask of the values defined in `NV_ENC_PREPROC_FLAGS`
- NV\_ENC\_CAPS\_ASYNC\_ENCODE\_SUPPORT*** Indicates support Async mode.  
0 : Async Encode mode not supported.  
1 : Async Encode mode supported.
- NV\_ENC\_CAPS\_MB\_NUM\_MAX*** Maximum MBs per frame supported.
- NV\_ENC\_CAPS\_MB\_PER\_SEC\_MAX*** Maximum aggregate throughput in MBs per sec.



***NV\_ENC\_CAPS\_SUPPORT\_YUV444\_ENCODE*** Indicates HW support for YUV444 mode encoding.

0 : YUV444 mode encoding not supported.

1 : YUV444 mode encoding supported.

***NV\_ENC\_CAPS\_SUPPORT\_LOSSLESS\_ENCODE*** Indicates HW support for lossless encoding.

0 : lossless encoding not supported.

1 : lossless encoding supported.

***NV\_ENC\_CAPS\_SUPPORT\_SAO*** Indicates HW support for Sample Adaptive Offset.

0 : SAO not supported.

1 : SAO encoding supported.

***NV\_ENC\_CAPS\_SUPPORT\_MEONLY\_MODE*** Indicates HW support for MEOnly Mode.

0 : MEOnly Mode not supported.

1 : MEOnly Mode supported.

***NV\_ENC\_CAPS\_EXPOSED\_COUNT*** Reserved - Not to be used by clients.

#### 4.1.2.3 enum NV\_ENC\_DEVICE\_TYPE

Encoder Device type

##### Enumerator:

***NV\_ENC\_DEVICE\_TYPE\_DIRECTX*** encode device type is a directx9 device

***NV\_ENC\_DEVICE\_TYPE\_CUDA*** encode device type is a cuda device

#### 4.1.2.4 enum NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_MODE

H.264 specific Adaptive Transform modes

##### Enumerator:

***NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_AUTOSELECT*** Adaptive Transform 8x8 mode is auto selected by the encoder driver

***NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_DISABLE*** Adaptive Transform 8x8 mode disabled

***NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_ENABLE*** Adaptive Transform 8x8 mode should be used

#### 4.1.2.5 enum NV\_ENC\_H264\_BDIRECT\_MODE

H.264 specific Bdirect modes

##### Enumerator:

***NV\_ENC\_H264\_BDIRECT\_MODE\_AUTOSELECT*** BDirect mode is auto selected by the encoder driver

***NV\_ENC\_H264\_BDIRECT\_MODE\_DISABLE*** Disable BDirect mode

***NV\_ENC\_H264\_BDIRECT\_MODE\_TEMPORAL*** Temporal BDirect mode

***NV\_ENC\_H264\_BDIRECT\_MODE\_SPATIAL*** Spatial BDirect mode

#### 4.1.2.6 enum NV\_ENC\_H264\_ENTROPY\_CODING\_MODE

H.264 entropy coding modes.

##### Enumerator:

- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_AUTOSELECT* Entropy coding mode is auto selected by the encoder driver
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_CABAC* Entropy coding mode is CABAC
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_CAVLC* Entropy coding mode is CAVLC

#### 4.1.2.7 enum NV\_ENC\_H264\_FMO\_MODE

H.264 specific FMO usage

##### Enumerator:

- NV\_ENC\_H264\_FMO\_AUTOSELECT* FMO usage is auto selected by the encoder driver
- NV\_ENC\_H264\_FMO\_ENABLE* Enable FMO
- NV\_ENC\_H264\_FMO\_DISABLE* Disble FMO

#### 4.1.2.8 enum NV\_ENC\_HEVC\_CUSIZE

HEVC CU SIZE

#### 4.1.2.9 enum NV\_ENC\_INPUT\_RESOURCE\_TYPE

Input Resource type

##### Enumerator:

- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_DIRECTX* input resource type is a directx9 surface
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDADEVICEPTR* input resource type is a cuda device pointer surface
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDAARRAY* input resource type is a cuda array surface

#### 4.1.2.10 enum NV\_ENC\_LEVEL

Encoding levels

#### 4.1.2.11 enum NV\_ENC\_MEMORY\_HEAP

Memory heap to allocate input and output buffers.

##### Enumerator:

- NV\_ENC\_MEMORY\_HEAP\_AUTOSELECT* Memory heap to be decided by the encoder driver based on the usage
- NV\_ENC\_MEMORY\_HEAP\_VID* Memory heap is in local video memory
- NV\_ENC\_MEMORY\_HEAP\_SYSMEM\_CACHED* Memory heap is in cached system memory
- NV\_ENC\_MEMORY\_HEAP\_SYSMEM\_UNCACHED* Memory heap is in uncached system memory

#### 4.1.2.12 enum NV\_ENC\_MV\_PRECISION

Motion vector precisions

##### Enumerator:

*NV\_ENC\_MV\_PRECISION\_DEFAULT* Driver selects QuarterPel motion vector precision by default  
*NV\_ENC\_MV\_PRECISION\_FULL\_PEL* FullPel motion vector precision  
*NV\_ENC\_MV\_PRECISION\_HALF\_PEL* HalfPel motion vector precision  
*NV\_ENC\_MV\_PRECISION\_QUARTER\_PEL* QuarterPel motion vector precision

#### 4.1.2.13 enum NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE

Input frame encode modes

##### Enumerator:

*NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_FRAME* Frame mode  
*NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_FIELD* Field mode  
*NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_MBAFF* MB adaptive frame/field

#### 4.1.2.14 enum NV\_ENC\_PARAMS\_RC\_MODE

Rate Control Modes

##### Enumerator:

*NV\_ENC\_PARAMS\_RC\_CONSTQP* Constant QP mode  
*NV\_ENC\_PARAMS\_RC\_VBR* Variable bitrate mode  
*NV\_ENC\_PARAMS\_RC\_CBR* Constant bitrate mode  
*NV\_ENC\_PARAMS\_RC\_VBR\_MINQP* Variable bitrate mode with MinQP  
*NV\_ENC\_PARAMS\_RC\_2\_PASS\_QUALITY* Multi pass encoding optimized for image quality and works only with low latency mode  
*NV\_ENC\_PARAMS\_RC\_2\_PASS\_FRAMESIZE\_CAP* Multi pass encoding optimized for maintaining frame size and works only with low latency mode  
*NV\_ENC\_PARAMS\_RC\_2\_PASS\_VBR* Multi pass VBR

#### 4.1.2.15 enum NV\_ENC\_PIC\_FLAGS

Encode Picture encode flags.

##### Enumerator:

*NV\_ENC\_PIC\_FLAG\_FORCEINTRA* Encode the current picture as an Intra picture  
*NV\_ENC\_PIC\_FLAG\_FORCEIDR* Encode the current picture as an IDR picture. This flag is only valid when Picture type decision is taken by the Encoder [*\_NV\_ENC\_INITIALIZE\_PARAMS::enablePTD* == 1].  
*NV\_ENC\_PIC\_FLAG\_OUTPUT\_SPSPPS* Write the sequence and picture header in encoded bitstream of the current picture  
*NV\_ENC\_PIC\_FLAG\_EOS* Indicates end of the input stream

**4.1.2.16 enum NV\_ENC\_PIC\_STRUCT**

Input picture structure

**Enumerator:**

*NV\_ENC\_PIC\_STRUCT\_FRAME* Progressive frame  
*NV\_ENC\_PIC\_STRUCT\_FIELD\_TOP\_BOTTOM* Field encoding top field first  
*NV\_ENC\_PIC\_STRUCT\_FIELD\_BOTTOM\_TOP* Field encoding bottom field first

**4.1.2.17 enum NV\_ENC\_PIC\_TYPE**

Input picture type

**Enumerator:**

*NV\_ENC\_PIC\_TYPE\_P* Forward predicted  
*NV\_ENC\_PIC\_TYPE\_B* Bi-directionally predicted picture  
*NV\_ENC\_PIC\_TYPE\_I* Intra predicted picture  
*NV\_ENC\_PIC\_TYPE\_IDR* IDR picture  
*NV\_ENC\_PIC\_TYPE\_BI* Bi-directionally predicted with only Intra MBs  
*NV\_ENC\_PIC\_TYPE\_SKIPPED* Picture is skipped  
*NV\_ENC\_PIC\_TYPE\_INTRA\_REFRESH* First picture in intra refresh cycle  
*NV\_ENC\_PIC\_TYPE\_UNKNOWN* Picture type unknown

**4.1.2.18 enum NV\_ENC\_STEREO\_PACKING\_MODE**

Stereo frame packing modes.

**Enumerator:**

*NV\_ENC\_STEREO\_PACKING\_MODE\_NONE* No Stereo packing required  
*NV\_ENC\_STEREO\_PACKING\_MODE\_CHECKERBOARD* Checkerboard mode for packing stereo frames  
*NV\_ENC\_STEREO\_PACKING\_MODE\_COLINTERLEAVE* Column Interleave mode for packing stereo frames  
*NV\_ENC\_STEREO\_PACKING\_MODE\_ROWINTERLEAVE* Row Interleave mode for packing stereo frames  
  
*NV\_ENC\_STEREO\_PACKING\_MODE\_SIDEBYSIDE* Side-by-side mode for packing stereo frames  
*NV\_ENC\_STEREO\_PACKING\_MODE\_TOPBOTTOM* Top-Bottom mode for packing stereo frames  
*NV\_ENC\_STEREO\_PACKING\_MODE\_FRAMESEQ* Frame Sequential mode for packing stereo frames

**4.1.2.19 enum NVENCSTATUS**

Error Codes

**Enumerator:**

*NV\_ENC\_SUCCESS* This indicates that API call returned with no errors.

- NV\_ENC\_ERR\_NO\_ENCODE\_DEVICE*** This indicates that no encode capable devices were detected.
- NV\_ENC\_ERR\_UNSUPPORTED\_DEVICE*** This indicates that devices pass by the client is not supported.
- NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE*** This indicates that the encoder device supplied by the client is not valid.
- NV\_ENC\_ERR\_INVALID\_DEVICE*** This indicates that device passed to the API call is invalid.
- NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST*** This indicates that device passed to the API call is no longer available and needs to be reinitialized. The clients need to destroy the current encoder session by freeing the allocated input output buffers and destroying the device and create a new encoding session.
- NV\_ENC\_ERR\_INVALID\_PTR*** This indicates that one or more of the pointers passed to the API call is invalid.
- NV\_ENC\_ERR\_INVALID\_EVENT*** This indicates that completion event passed in [NvEncEncodePicture\(\)](#) call is invalid.
- NV\_ENC\_ERR\_INVALID\_PARAM*** This indicates that one or more of the parameter passed to the API call is invalid.
- NV\_ENC\_ERR\_INVALID\_CALL*** This indicates that an API call was made in wrong sequence/order.
- NV\_ENC\_ERR\_OUT\_OF\_MEMORY*** This indicates that the API call failed because it was unable to allocate enough memory to perform the requested operation.
- NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED*** This indicates that the encoder has not been initialized with [NvEncInitializeEncoder\(\)](#) or that initialization has failed. The client cannot allocate input or output buffers or do any encoding related operation before successfully initializing the encoder.
- NV\_ENC\_ERR\_UNSUPPORTED\_PARAM*** This indicates that an unsupported parameter was passed by the client.
- NV\_ENC\_ERR\_LOCK\_BUSY*** This indicates that the [NvEncLockBitstream\(\)](#) failed to lock the output buffer. This happens when the client makes a non blocking lock call to access the output bitstream by passing `NV_ENC_LOCK_BITSTREAM::doNotWait` flag. This is not a fatal error and client should retry the same operation after few milliseconds.
- NV\_ENC\_ERR\_NOT\_ENOUGH\_BUFFER*** This indicates that the size of the user buffer passed by the client is insufficient for the requested operation.
- NV\_ENC\_ERR\_INVALID\_VERSION*** This indicates that an invalid struct version was used by the client.
- NV\_ENC\_ERR\_MAP\_FAILED*** This indicates that [NvEncMapInputResource\(\)](#) API failed to map the client provided input resource.
- NV\_ENC\_ERR\_NEED\_MORE\_INPUT*** This indicates encode driver requires more input buffers to produce an output bitstream. If this error is returned from [NvEncEncodePicture\(\)](#) API, this is not a fatal error. If the client is encoding with B frames then, [NvEncEncodePicture\(\)](#) API might be buffering the input frame for re-ordering.
- A client operating in synchronous mode cannot call [NvEncLockBitstream\(\)](#) API on the output bitstream buffer if [NvEncEncodePicture\(\)](#) returned the `NV_ENC_ERR_NEED_MORE_INPUT` error code. The client must continue providing input frames until encode driver returns `NV_ENC_SUCCESS`. After receiving `NV_ENC_SUCCESS` status the client can call [NvEncLockBitstream\(\)](#) API on the output buffers in the same order in which it has called [NvEncEncodePicture\(\)](#).
- NV\_ENC\_ERR\_ENCODER\_BUSY*** This indicates that the HW encoder is busy encoding and is unable to encode the input. The client should call [NvEncEncodePicture\(\)](#) again after few milliseconds.
- NV\_ENC\_ERR\_EVENT\_NOT\_REGISTERD*** This indicates that the completion event passed in [NvEncEncodePicture\(\)](#) API has not been registered with encoder driver using [NvEncRegisterAsyncEvent\(\)](#).
- NV\_ENC\_ERR\_GENERIC*** This indicates that an unknown internal error has occurred.
- NV\_ENC\_ERR\_INCOMPATIBLE\_CLIENT\_KEY*** This indicates that the client is attempting to use a feature that is not available for the license type for the current system.

***NV\_ENC\_ERR\_UNIMPLEMENTED*** This indicates that the client is attempting to use a feature that is not implemented for the current version.

***NV\_ENC\_ERR\_RESOURCE\_REGISTER\_FAILED*** This indicates that the [NvEncRegisterResource](#) API failed to register the resource.

***NV\_ENC\_ERR\_RESOURCE\_NOT\_REGISTERED*** This indicates that the client is attempting to unregister a resource that has not been successfully registered.

***NV\_ENC\_ERR\_RESOURCE\_NOT\_MAPPED*** This indicates that the client is attempting to unmap a resource that has not been successfully mapped.

## 4.2 NvEncodeAPI Functions

### Functions

- **NVENCSTATUS** NVENCAPI **NvEncOpenEncodeSession** (void \*device, uint32\_t deviceType, void \*\*encoder)  
*Opens an encoding session.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodeGUIDCount** (void \*encoder, uint32\_t \*encodeGUIDCount)  
*Retrieves the number of supported encode GUIDs.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodeGUIDs** (void \*encoder, GUID \*GUIDs, uint32\_t guidArraySize, uint32\_t \*GUIDCount)  
*Retrieves an array of supported encoder codec GUIDs.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodeProfileGUIDCount** (void \*encoder, GUID encodeGUID, uint32\_t \*encodeProfileGUIDCount)  
*Retrieves the number of supported profile GUIDs.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodeProfileGUIDs** (void \*encoder, GUID encodeGUID, GUID \*profileGUIDs, uint32\_t guidArraySize, uint32\_t \*GUIDCount)  
*Retrieves an array of supported encode profile GUIDs.*
- **NVENCSTATUS** NVENCAPI **NvEncGetInputFormatCount** (void \*encoder, GUID encodeGUID, uint32\_t \*inputFmtCount)  
*Retrieve the number of supported Input formats.*
- **NVENCSTATUS** NVENCAPI **NvEncGetInputFormats** (void \*encoder, GUID encodeGUID, NV\_ENC\_BUFFER\_FORMAT \*inputFmts, uint32\_t inputFmtArraySize, uint32\_t \*inputFmtCount)  
*Retrieves an array of supported Input formats.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodeCaps** (void \*encoder, GUID encodeGUID, NV\_ENC\_CAPS\_PARAM \*capsParam, int \*capsVal)  
*Retrieves the capability value for a specified encoder attribute.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodePresetCount** (void \*encoder, GUID encodeGUID, uint32\_t \*encodePresetGUIDCount)  
*Retrieves the number of supported preset GUIDs.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodePresetGUIDs** (void \*encoder, GUID encodeGUID, GUID \*presetGUIDs, uint32\_t guidArraySize, uint32\_t \*encodePresetGUIDCount)  
*Receives an array of supported encoder preset GUIDs.*
- **NVENCSTATUS** NVENCAPI **NvEncGetEncodePresetConfig** (void \*encoder, GUID encodeGUID, GUID presetGUID, NV\_ENC\_PRESET\_CONFIG \*presetConfig)  
*Returns a preset config structure supported for given preset GUID.*
- **NVENCSTATUS** NVENCAPI **NvEncInitializeEncoder** (void \*encoder, NV\_ENC\_INITIALIZE\_PARAMS \*createEncodeParams)  
*Initialize the encoder.*

- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncCreateInputBuffer](#) (void \*encoder, [NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER](#) \*createInputBufferParams)  
*Allocates Input buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncDestroyInputBuffer](#) (void \*encoder, [NV\\_ENC\\_INPUT\\_PTR](#) inputBuffer)  
*Release an input buffers.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncCreateBitstreamBuffer](#) (void \*encoder, [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER](#) \*createBitstreamBufferParams)  
*Allocates an output bitstream buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncDestroyBitstreamBuffer](#) (void \*encoder, [NV\\_ENC\\_OUTPUT\\_PTR](#) bitstreamBuffer)  
*Release a bitstream buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncEncodePicture](#) (void \*encoder, [NV\\_ENC\\_PIC\\_PARAMS](#) \*encodePicParams)  
*Submit an input picture for encoding.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncLockBitstream](#) (void \*encoder, [NV\\_ENC\\_LOCK\\_BITSTREAM](#) \*lockBitstreamBufferParams)  
*Lock output bitstream buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncUnlockBitstream](#) (void \*encoder, [NV\\_ENC\\_OUTPUT\\_PTR](#) bitstreamBuffer)  
*Unlock the output bitstream buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncLockInputBuffer](#) (void \*encoder, [NV\\_ENC\\_LOCK\\_INPUT\\_BUFFER](#) \*lockInputBufferParams)  
*Locks an input buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncUnlockInputBuffer](#) (void \*encoder, [NV\\_ENC\\_INPUT\\_PTR](#) inputBuffer)  
*Unlocks the input buffer.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncGetEncodeStats](#) (void \*encoder, [NV\\_ENC\\_STAT](#) \*encodeStats)  
*Get encoding statistics.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncGetSequenceParams](#) (void \*encoder, [NV\\_ENC\\_SEQUENCE\\_PARAM\\_PAYLOAD](#) \*sequenceParamPayload)  
*Get encoded sequence and picture header.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncRegisterAsyncEvent](#) (void \*encoder, [NV\\_ENC\\_EVENT\\_PARAMS](#) \*eventParams)  
*Register event for notification to encoding completion.*
- [NVENCSTATUS](#) [NVENCAPI](#) [NvEncUnregisterAsyncEvent](#) (void \*encoder, [NV\\_ENC\\_EVENT\\_PARAMS](#) \*eventParams)  
*Unregister completion event.*



- **NVENCSTATUS** NVENCAPI [NvEncMapInputResource](#) (void \*encoder, NV\_ENC\_MAP\_INPUT\_RESOURCE \*mapInputResParams)  
*Map an externally created input resource pointer for encoding.*
- **NVENCSTATUS** NVENCAPI [NvEncUnmapInputResource](#) (void \*encoder, NV\_ENC\_INPUT\_PTR mapped-InputBuffer)  
*UnMaps a NV\_ENC\_INPUT\_PTR which was mapped for encoding.*
- **NVENCSTATUS** NVENCAPI [NvEncDestroyEncoder](#) (void \*encoder)  
*Destroy Encoding Session.*
- **NVENCSTATUS** NVENCAPI [NvEncInvalidateRefFrames](#) (void \*encoder, uint64\_t invalidRefFrameTimeStamp)  
*Invalidate reference frames.*
- **NVENCSTATUS** NVENCAPI [NvEncOpenEncodeSessionEx](#) (NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS \*openSessionExParams, void \*\*encoder)  
*Opens an encoding session.*
- **NVENCSTATUS** NVENCAPI [NvEncRegisterResource](#) (void \*encoder, NV\_ENC\_REGISTER\_RESOURCE \*registerResParams)  
*Registers a resource with the Nvidia Video Encoder Interface.*
- **NVENCSTATUS** NVENCAPI [NvEncUnregisterResource](#) (void \*encoder, NV\_ENC\_REGISTERED\_PTR registeredResource)  
*Unregisters a resource previously registered with the Nvidia Video Encoder Interface.*
- **NVENCSTATUS** NVENCAPI [NvEncReconfigureEncoder](#) (void \*encoder, NV\_ENC\_RECONFIGURE\_PARAMS \*reInitEncodeParams)  
*Reconfigure an existing encoding session.*
- **NVENCSTATUS** NVENCAPI [NvEncCreateMVBuffer](#) (void \*encoder, NV\_ENC\_CREATE\_MV\_BUFFER \*createMVBufferParams)  
*Allocates output MV buffer for ME only mode.*
- **NVENCSTATUS** NVENCAPI [NvEncDestroyMVBuffer](#) (void \*encoder, NV\_ENC\_OUTPUT\_PTR MVBuffer)  
*Release an output MV buffer for ME only mode.*
- **NVENCSTATUS** NVENCAPI [NvEncRunMotionEstimationOnly](#) (void \*encoder, NV\_ENC\_MEONLY\_PARAMS \*MEOnlyParams)  
*Submit an input picture and reference frame for motion estimation in ME only mode.*
- **NVENCSTATUS** NVENCAPI [NvEncodeAPICreateInstance](#) (NV\_ENCODE\_API\_FUNCTION\_LIST \*functionList)

## 4.2.1 Function Documentation

### 4.2.1.1 NVENCSTATUS NVENCAPI NvEncCreateBitstreamBuffer (void \* *encoder*, NV\_ENC\_CREATE\_BITSTREAM\_BUFFER \* *createBitstreamBufferParams*)

This function is used to allocate an output bitstream buffer and returns a NV\_ENC\_OUTPUT\_PTR to bitstream buffer to the client in the [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER::bitstreamBuffer](#) field. The client can only call this function after the encoder session has been initialized using [NvEncInitializeEncoder\(\)](#) API. The minimum number of output buffers allocated by the client must be at least 4 more than the number of B B frames being used for encoding. The client can only access the output bitstream data by locking the `bitstreamBuffer` using the [NvEncLockBitstream\(\)](#) function.

#### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *createBitstreamBufferParams* Pointer [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER](#) for details.

#### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

### 4.2.1.2 NVENCSTATUS NVENCAPI NvEncCreateInputBuffer (void \* *encoder*, NV\_ENC\_CREATE\_INPUT\_BUFFER \* *createInputBufferParams*)

This function is used to allocate an input buffer. The client must enumerate the input buffer format before allocating the input buffer resources. The NV\_ENC\_INPUT\_PTR returned by the NvEncodeAPI interface in the [NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER::inputBuffer](#) field can be directly used in [NvEncEncodePicture\(\)](#) API. The number of input buffers to be allocated by the client must be at least 4 more than the number of B frames being used for encoding.

#### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *createInputBufferParams* Pointer to the [NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER](#) structure.

#### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.3 NVENCSTATUS NVENCAPI NvEncCreateMVBuffer (void \* *encoder*, NV\_ENC\_CREATE\_MV\_BUFFER \* *createMVBufferParams*)

This function is used to allocate an output MV buffer. The size of the MVBuffer is dependent on the frame height and width of the last NvEncCreateInputBuffer call. The NV\_ENC\_OUTPUT\_PTR returned by the NvEncodeAPI interface in the NV\_ENC\_CREATE\_MV\_BUFFER::MVBuffer field can be used in NvEncRunMotionEstimationOnly() API.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *createMVBufferParams* Pointer to the NV\_ENC\_CREATE\_MV\_BUFFER structure.

##### Returns:

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_GENERIC

#### 4.2.1.4 NVENCSTATUS NVENCAPI NvEncDestroyBitstreamBuffer (void \* *encoder*, NV\_ENC\_OUTPUT\_PTR *bitstreamBuffer*)

This function is used to release the output bitstream buffer allocated using the NvEncCreateBitstreamBuffer() function. The client must release the output bitstreamBuffer using this function before destroying the encoder session.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *bitstreamBuffer* Pointer to the bitstream buffer being released.

##### Returns:

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
 NV\_ENC\_ERR\_GENERIC

#### 4.2.1.5 NVENCSTATUS NVENCAPI NvEncDestroyEncoder (void \* *encoder*)

Destroys the encoder session previously created using NvEncOpenEncodeSession() function. The client must flush the encoder before freeing any resources. In order to flush the encoder the client must pass a NULL encode picture

packet and either wait for the [NvEncEncodePicture\(\)](#) function to return in synchronous mode or wait for the flush event to be signaled by the encoder in asynchronous mode. The client must free all the input and output resources created using the NvEncodeAPI interface before destroying the encoder. If the client is operating in asynchronous mode, it must also unregister the completion events previously registered.

**Parameters:**

← *encoder* Pointer to the NvEncodeAPI interface.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

**4.2.1.6 NVENCSTATUS NVENCAPI NvEncDestroyInputBuffer (void \* *encoder*, NV\_ENC\_INPUT\_PTR *inputBuffer*)**

This function is used to free an input buffer. If the client has allocated any input buffer using [NvEncCreateInputBuffer\(\)](#) API, it must free those input buffers by calling this function. The client must release the input buffers before destroying the encoder using [NvEncDestroyEncoder\(\)](#) API.

**Parameters:**

← *encoder* Pointer to the NvEncodeAPI interface.  
 ← *inputBuffer* Pointer to the input buffer to be released.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

**4.2.1.7 NVENCSTATUS NVENCAPI NvEncDestroyMVBuffer (void \* *encoder*, NV\_ENC\_OUTPUT\_PTR *MVBuffer*)**

This function is used to release the output MV buffer allocated using the :: [NvEncCreateMVBuffer\(\)](#) function. The client must release the output MVBuffer using this function before destroying the encoder session.

**Parameters:**

← *encoder* Pointer to the NvEncodeAPI interface.

← *MVBuffer* Pointer to the MVBuffer being released.

#### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.8 NVENCSTATUS NVENCAPI NvEncEncodePicture (void \* *encoder*, NV\_ENC\_PIC\_PARAMS \* *encodePicParams*)

This function is used to submit an input picture buffer for encoding. The encoding parameters are passed using \**encodePicParams* which is a pointer to the [\\_NV\\_ENC\\_PIC\\_PARAMS](#) structure.

If the client has set NV\_ENC\_INITIALIZE\_PARAMS::enablePTD to 0, then it must send a valid value for the following fields.

- NV\_ENC\_PIC\_PARAMS::pictureType
- NV\_ENC\_PIC\_PARAMS\_H264::displayPOCSyntax (H264 only)
- NV\_ENC\_PIC\_PARAMS\_H264::frameNumSyntax(H264 only)
- NV\_ENC\_PIC\_PARAMS\_H264::refPicFlag(H264 only)

#### Asynchronous Encoding

If the client has enabled asynchronous mode of encoding by setting NV\_ENC\_INITIALIZE\_PARAMS::enableEncodeAsync to 1 in the [NvEncInitializeEncoder\(\)](#) API ,then the client must send a valid NV\_ENC\_PIC\_PARAMS::completionEvent. In case of asynchronous mode of operation, client can queue the [NvEncEncodePicture\(\)](#) API commands from the main thread and then queue output buffers to be processed to a secondary worker thread. Before the locking the output buffers in the secondary thread , the client must wait on NV\_ENC\_PIC\_PARAMS::completionEvent it has queued in [NvEncEncodePicture\(\)](#) API call. The client must always process completion event and the output buffer in the same order in which they have been submitted for encoding. The NvEncodeAPI interface is responsible for any re-ordering required for B frames and will always ensure that encoded bitstream data is written in the same order in which output buffer is submitted.

The below example shows how asynchronous encoding in case of 1 B frames

-----  
 Suppose the client allocated 4 input buffers(I1,I2..), 4 output buffers(O1,O2..) and 4 completion events(E1, E2, ...). The NvEncodeAPI interface will need to keep a copy of the input buffers for re-ordering and it allocates following internal buffers (NvI1, NvI2...). These internal buffers are managed by NvEncodeAPI and the client is not responsible for the allocating or freeing the memory of the internal buffers.

a) The client main thread will queue the following encode frame calls.  
 Note the picture type is unknown to the client, the decision is being taken by NvEncodeAPI interface. The client should pass ::NV\_ENC\_PIC\_PARAMS parameter consisting of allocated input buffer, output buffer and output events in successive ::NvEncEncodePicture() API calls along with other required encode picture params.

For example:

```
1st EncodePicture parameters - (I1, O1, E1)
2nd EncodePicture parameters - (I2, O2, E2)
3rd EncodePicture parameters - (I3, O3, E3)
```

b) NvEncodeAPI SW will receive the following encode Commands from the client. The left side shows input from client in the form (Input buffer, Output Buffer, Output Event). The right hand side shows a possible picture type decision take by the NvEncodeAPI interface.

```
(I1, O1, E1)    ---P1 Frame
(I2, O2, E2)    ---B2 Frame
(I3, O3, E3)    ---P3 Frame
```

c) NvEncodeAPI interface will make a copy of the input buffers to its internal buffers for re-ordering. These copies are done as part of nvEncEncodePicture function call from the client and NvEncodeAPI interface is responsible for synchronization of copy operation with the actual encoding operation.

```
I1 --> NvI1
I2 --> NvI2
I3 --> NvI3
```

d) After returning from ::NvEncEncodePicture() call , the client must queue the output bitstream processing work to the secondary thread. The output bitstream processing for asynchronous mode consist of first waiting on completion event(E1, E2..) and then locking the output bitstream buffer(O1, O2..) for reading the encoded data. The work queued to the secondary thread by the client is in the following order

```
(I1, O1, E1)
(I2, O2, E2)
(I3, O3, E3)
```

Note they are in the same order in which client calls ::NvEncEncodePicture() API in \p step a).

e) NvEncodeAPI interface will do the re-ordering such that Encoder HW will receive the following encode commands:

```
(NvI1, O1, E1)    ---P1 Frame
(NvI3, O2, E2)    ---P3 Frame
(NvI2, O3, E3)    ---B2 frame
```

f) After the encoding operations are completed, the events will be signalled by NvEncodeAPI interface in the following order :

```
(O1, E1) ---P1 Frame ,output bitstream copied to O1 and event E1 signalled.
(O2, E2) ---P3 Frame ,output bitstream copied to O2 and event E2 signalled.
(O3, E3) ---B2 Frame ,output bitstream copied to O3 and event E3 signalled.
```

g) The client must lock the bitstream data using ::NvEncLockBitstream() API in the order O1,O2,O3 to read the encoded data, after waiting for the events to be signalled in the same order i.e E1, E2 and E3.The output processing is done in the secondary thread in the following order:

```
Waits on E1, copies encoded bitstream from O1
Waits on E2, copies encoded bitstream from O2
Waits on E3, copies encoded bitstream from O3
```

-Note the client will receive the events signalling and output buffer in the same order in which they have submitted for encoding.

-Note the LockBitstream will have picture type field which will notify the output picture type to the clients.

-Note the input, output buffer and the output completion event are free to be reused once NvEncodeAPI interfaced has signalled the event and the client has copied the data from the output buffer.

## Synchronous Encoding

The client can enable synchronous mode of encoding by setting NV\_ENC\_INITIALIZE\_PARAMS::enableEncodeAsync to 0 in NvEncInitializeEncoder() API. The NvEncodeAPI interface may return NV\_ENC\_ERR\_NEED\_MORE\_INPUT error code for some NvEncEncodePicture() API calls when NV\_ENC\_INITIALIZE\_PARAMS::enablePTD is set to 1, but the client must not treat it as a fatal error. The NvEncodeAPI interface might not be able to submit an input picture buffer for encoding immediately due to

re-ordering for B frames. The NvEncodeAPI interface cannot submit the input picture which is decided to be encoded as B frame as it waits for backward reference from temporally subsequent frames. This input picture is buffered internally and waits for more input picture to arrive. The client must not call `NvEncLockBitstream()` API on the output buffers whose `NvEncEncodePicture()` API returns `NV_ENC_ERR_NEED_MORE_INPUT`. The client must wait for the NvEncodeAPI interface to return `NV_ENC_SUCCESS` before locking the output bitstreams to read the encoded bitstream data. The following example explains the scenario with synchronous encoding with 2 B frames.

The below example shows how synchronous encoding works in case of 1 B frames

-----  
 Suppose the client allocated 4 input buffers(I1,I2..), 4 output buffers(O1,O2..) and 4 completion events(E1, E2, ...). The NvEncodeAPI interface will need to keep a copy of the input buffers for re-ordering and it allocates following internal buffers (NvI1, NvI2...). These internal buffers are managed by NvEncodeAPI and the client is not responsible for the allocating or freeing the memory of the internal buffers.

The client calls `::NvEncEncodePicture()` API with input buffer I1 and output buffer O1. The NvEncodeAPI decides to encode I1 as P frame and submits it to encoder HW and returns `::NV_ENC_SUCCESS`. The client can now read the encoded data by locking the output O1 by calling `NvEncLockBitstream` API.

The client calls `::NvEncEncodePicture()` API with input buffer I2 and output buffer O2. The NvEncodeAPI decides to encode I2 as B frame and buffers I2 by copying it to internal buffer and returns `::NV_ENC_ERR_NEED_MORE_INPUT`. The error is not fatal and it notifies client that it cannot read the encoded data by locking the output O2 by calling `::NvEncLockBitstream()` API without submitting more work to the NvEncodeAPI interface.

The client calls `::NvEncEncodePicture()` with input buffer I3 and output buffer O3. The NvEncodeAPI decides to encode I3 as P frame and it first submits I3 for encoding which will be used as backward reference frame for I2. The NvEncodeAPI then submits I2 for encoding and returns `::NV_ENC_SUCCESS`. Both the submission are part of the same `::NvEncEncodePicture()` function call. The client can now read the encoded data for both the frames by locking the output O2 followed by O3 ,by calling `::NvEncLockBitstream()` API.

The client must always lock the output in the same order in which it has submitted to receive the encoded bitstream in correct encoding order.

#### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *encodePicParams* Pointer to the `_NV_ENC_PIC_PARAMS` structure.

#### Returns:

`NV_ENC_SUCCESS`  
`NV_ENC_ERR_INVALID_PTR`  
`NV_ENC_ERR_INVALID_ENCODERDEVICE`  
`NV_ENC_ERR_DEVICE_NOT_EXIST`  
`NV_ENC_ERR_UNSUPPORTED_PARAM`  
`NV_ENC_ERR_OUT_OF_MEMORY`  
`NV_ENC_ERR_INVALID_PARAM`  
`NV_ENC_ERR_INVALID_VERSION`  
`NV_ENC_ERR_ENCODER_BUSY`  
`NV_ENC_ERR_NEED_MORE_INPUT`  
`NV_ENC_ERR_ENCODER_NOT_INITIALIZED`  
`NV_ENC_ERR_GENERIC`

#### 4.2.1.9 NVENCSTATUS NVENCAPI NvEncGetEncodeCaps (void \* *encoder*, GUID *encodeGUID*, NV\_ENC\_CAPS\_PARAM \* *capsParam*, int \* *capsVal*)

The function returns the capability value for a given encoder attribute. The client must validate the encodeGUID using [NvEncGetEncodeGUIDs\(\)](#) API before calling this function. The encoder attribute being queried are enumerated in [NV\\_ENC\\_CAPS\\_PARAM](#) enum.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *encodeGUID* Encode GUID, corresponding to which the capability attribute is to be retrieved.
- ← *capsParam* Used to specify attribute being queried. Refer [NV\\_ENC\\_CAPS\\_PARAM](#) for more details.
- *capsVal* The value corresponding to the capability attribute being queried.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.10 NVENCSTATUS NVENCAPI NvEncGetEncodeGUIDCount (void \* *encoder*, uint32\_t \* *encodeGUIDCount*)

The function returns the number of codec guids supported by the NvEncodeAPI interface.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- *encodeGUIDCount* Number of supported encode GUIDs.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.11 NVENCSTATUS NVENCAPI NvEncGetEncodeGUIDs (void \* *encoder*, GUID \* *GUIDs*, uint32\_t *guidArraySize*, uint32\_t \* *GUIDCount*)

The function returns an array of codec guids supported by the NvEncodeAPI interface. The client must allocate an array where the NvEncodeAPI interface can fill the supported guids and pass the pointer in \*GUIDs parameter. The size of the array can be determined by using [NvEncGetEncodeGUIDCount\(\)](#) API. The Nvidia Encoding interface returns the number of codec guids it has actually filled in the guid array in the GUIDCount parameter.



**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *guidArraySize* Number of GUIDs to retrieved. Should be set to the number retrieved using [NvEncGetEncodeGUIDCount](#).
- *GUIDs* Array of supported Encode GUIDs.
- *GUIDCount* Number of supported Encode GUIDs.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.12 NVENCSTATUS NVENCAPI NvEncGetEncodePresetConfig (void \* *encoder*, GUID *encodeGUID*, GUID *presetGUID*, NV\_ENC\_PRESET\_CONFIG \* *presetConfig*)

The function returns a preset config structure for a given preset guid. Before using this function the client must enumerate the preset guides available for a given codec. The preset config structure can be modified by the client depending upon its use case and can be then used to initialize the encoder using [NvEncInitializeEncoder\(\)](#) API. The client can use this function only if it wants to modify the NvEncodeAPI preset configuration, otherwise it can directly use the preset guid.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *encodeGUID* Encode [GUID](#), corresponding to which the list of supported presets is to be retrieved.
- ← *presetGUID* Preset [GUID](#), corresponding to which the Encoding configurations is to be retrieved.
- *presetConfig* The requested Preset Encoder Attribute set. Refer [\\_NV\\_ENC\\_CONFIG](#) for more details.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.13 NVENCSTATUS NVENCAPI NvEncGetEncodePresetCount (void \* *encoder*, GUID *encodeGUID*, uint32\_t \* *encodePresetGUIDCount*)

The function returns the number of preset GUIDs available for a given codec. The client must validate the codec guid using [NvEncGetEncodeGUIDs\(\)](#) API before calling this function.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *encodeGUID* Encode [GUID](#), corresponding to which the number of supported presets is to be retrieved.
- *encodePresetGUIDCount* Receives the number of supported preset GUIDs.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.14 NVENCSTATUS NVENC\_API NvEncGetEncodePresetGUIDs (void \* *encoder*, GUID *encodeGUID*, GUID \* *presetGUIDs*, uint32\_t *guidArraySize*, uint32\_t \* *encodePresetGUIDCount*)

The function returns an array of encode preset guides available for a given codec. The client can directly use one of the preset guides based upon the use case or target device. The preset guide chosen can be directly used in NV\_ENC\_INITIALIZE\_PARAMS::presetGUID parameter to [NvEncEncodePicture\(\)](#) API. Alternately client can also use the preset guide to retrieve the encoding config parameters being used by NvEncodeAPI interface for that given preset, using [NvEncGetEncodePresetConfig\(\)](#) API. It can then modify preset config parameters as per its use case and send it to NvEncodeAPI interface as part of NV\_ENC\_INITIALIZE\_PARAMS::encodeConfig parameter for [NvEncInitializeEncoder\(\)](#) API.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *encodeGUID* Encode [GUID](#), corresponding to which the list of supported presets is to be retrieved.
- ← *guidArraySize* Size of array of preset guides passed in `preset` GUIDs
- *presetGUIDs* Array of supported Encode preset GUIDs from the NvEncodeAPI interface to client.
- *encodePresetGUIDCount* Receives the number of preset GUIDs returned by the NvEncodeAPI interface.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.15 NVENCSTATUS NVENC\_API NvEncGetEncodeProfileGUIDCount (void \* *encoder*, GUID *encodeGUID*, uint32\_t \* *encodeProfileGUIDCount*)

The function returns the number of profile GUIDs supported for a given codec. The client must first enumerate the codec guides supported by the NvEncodeAPI interface. After determining the codec guid, it can query the NvEncodeAPI interface to determine the number of profile guides supported for a particular codec guid.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *encodeGUID* The codec guid for which the profile guids are being enumerated.
- *encodeProfileGUIDCount* Number of encode profiles supported for the given encodeGUID.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.16 NVENCSTATUS NVENCAPI NvEncGetEncodeProfileGUIDs (void \* *encoder*, GUID *encodeGUID*, GUID \* *profileGUIDs*, uint32\_t *guidArraySize*, uint32\_t \* *GUIDCount*)

The function returns an array of supported profile guids for a particular codec guid. The client must allocate an array where the NvEncodeAPI interface can populate the profile guids. The client can determine the array size using [NvEncGetEncodeProfileGUIDCount\(\)](#) API. The client must also validate that the NvEncodeAPI interface supports the GUID the client wants to pass as *encodeGUID* parameter.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *encodeGUID* The encode guid whose profile guids are being enumerated.
- ← *guidArraySize* Number of GUIDs to be retrieved. Should be set to the number retrieved using [NvEncGetEncodeProfileGUIDCount](#).
- *profileGUIDs* Array of supported Encode Profile GUIDs
- *GUIDCount* Number of valid encode profile GUIDs in *profileGUIDs* array.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.17 NVENCSTATUS NVENCAPI NvEncGetEncodeStats (void \* *encoder*, NV\_ENC\_STAT \* *encodeStats*)

This function is used to retrieve the encoding statistics. This API is not supported when encode device type is CUDA.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.

↔ *encodeStats* Pointer to the `_NV_ENC_STAT` structure.

**Returns:**

`NV_ENC_SUCCESS`  
`NV_ENC_ERR_INVALID_PTR`  
`NV_ENC_ERR_INVALID_ENCODERDEVICE`  
`NV_ENC_ERR_DEVICE_NOT_EXIST`  
`NV_ENC_ERR_UNSUPPORTED_PARAM`  
`NV_ENC_ERR_OUT_OF_MEMORY`  
`NV_ENC_ERR_INVALID_PARAM`  
`NV_ENC_ERR_ENCODER_NOT_INITIALIZED`  
`NV_ENC_ERR_GENERIC`

**4.2.1.18 NVENCSTATUS NVENCAPI NvEncGetInputFormatCount (void \* *encoder*, GUID *encodeGUID*, uint32\_t \* *inputFmtCount*)**

The function returns the number of supported input formats. The client must query the NvEncodeAPI interface to determine the supported input formats before creating the input surfaces.

**Parameters:**

← *encoder* Pointer to the NvEncodeAPI interface.  
 ← *encodeGUID* Encode `GUID`, corresponding to which the number of supported input formats is to be retrieved.  
 → *inputFmtCount* Number of input formats supported for specified Encode `GUID`.

**Returns:**

`NV_ENC_SUCCESS`  
`NV_ENC_ERR_INVALID_PTR`  
`NV_ENC_ERR_INVALID_ENCODERDEVICE`  
`NV_ENC_ERR_DEVICE_NOT_EXIST`  
`NV_ENC_ERR_UNSUPPORTED_PARAM`  
`NV_ENC_ERR_OUT_OF_MEMORY`  
`NV_ENC_ERR_INVALID_PARAM`  
`NV_ENC_ERR_GENERIC`

**4.2.1.19 NVENCSTATUS NVENCAPI NvEncGetInputFormats (void \* *encoder*, GUID *encodeGUID*, NV\_ENC\_BUFFER\_FORMAT \* *inputFmts*, uint32\_t *inputFmtArraySize*, uint32\_t \* *inputFmtCount*)**

Returns an array of supported input formats. The client must use the input format to create input surface using `NvEnc-CreateInputBuffer()` API.

**Parameters:**

← *encoder* Pointer to the NvEncodeAPI interface.  
 ← *encodeGUID* Encode `GUID`, corresponding to which the number of supported input formats is to be retrieved.  
 ← *inputFmtArraySize* Size input format count array passed in `inputFmts`.  
 → *inputFmts* Array of input formats supported for this Encode `GUID`.

→ *inputFmtCount* The number of valid input format types returned by the NvEncodeAPI interface in *inputFmts* array.

**Returns:**

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_GENERIC

#### 4.2.1.20 NVENCSTATUS NVENCAPI NvEncGetSequenceParams (void \* *encoder*, NV\_ENC\_SEQUENCE\_PARAM\_PAYLOAD \* *sequenceParamPayload*)

This function can be used to retrieve the sequence and picture header out of band. The client must call this function only after the encoder has been initialized using [NvEncInitializeEncoder\(\)](#) function. The client must allocate the memory where the NvEncodeAPI interface can copy the bitstream header and pass the pointer to the memory in NV\_ENC\_SEQUENCE\_PARAM\_PAYLOAD::spsppsBuffer. The size of buffer is passed in the field NV\_ENC\_SEQUENCE\_PARAM\_PAYLOAD::inBufferSize. The NvEncodeAPI interface will copy the bitstream header payload and returns the actual size of the bitstream header in the field NV\_ENC\_SEQUENCE\_PARAM\_PAYLOAD::outSPSPSPayloadSize. The client must call [NvEncGetSequenceParams\(\)](#) function from the same thread which is being used to call [NvEncEncodePicture\(\)](#) function.

**Parameters:**

← *encoder* Pointer to the NvEncodeAPI interface.  
 ↔ *sequenceParamPayload* Pointer to the [\\_NV\\_ENC\\_SEQUENCE\\_PARAM\\_PAYLOAD](#) structure.

**Returns:**

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
 NV\_ENC\_ERR\_GENERIC

#### 4.2.1.21 NVENCSTATUS NVENCAPI NvEncInitializeEncoder (void \* *encoder*, NV\_ENC\_INITIALIZE\_PARAMS \* *createEncodeParams*)

This API must be used to initialize the encoder. The initialization parameter is passed using *\*createEncodeParams*. The client must send the following fields of the [\\_NV\\_ENC\\_INITIALIZE\\_PARAMS](#) structure with a valid value.

- NV\_ENC\_INITIALIZE\_PARAMS::encodeGUID

- NV\_ENC\_INITIALIZE\_PARAMS::encodeWidth
- NV\_ENC\_INITIALIZE\_PARAMS::encodeHeight

The client can pass a preset guid directly to the NvEncodeAPI interface using NV\_ENC\_INITIALIZE\_PARAMS::presetGUID field. If the client doesn't pass NV\_ENC\_INITIALIZE\_PARAMS::encodeConfig structure, the codec specific parameters will be selected based on the preset guid. The preset guid must have been validated by the client using [NvEncGetEncodePresetGUIDs\(\)](#) API. If the client passes a custom [\\_NV\\_ENC\\_CONFIG](#) structure through NV\_ENC\_INITIALIZE\_PARAMS::encodeConfig , it will override the codec specific parameters based on the preset guid. It is recommended that even if the client passes a custom config, it should also send a preset guid. In this case, the preset guid passed by the client will not override any of the custom config parameters programmed by the client, it is only used as a hint by the NvEncodeAPI interface to determine certain encoder parameters which are not exposed to the client.

There are two modes of operation for the encoder namely:

- Asynchronous mode
- Synchronous mode

The client can select asynchronous or synchronous mode by setting the enableEncodeAsync field in [\\_NV\\_ENC\\_INITIALIZE\\_PARAMS](#) to 1 or 0 respectively.

#### Asynchronous mode of operation:

The Asynchronous mode can be enabled by setting NV\_ENC\_INITIALIZE\_PARAMS::enableEncodeAsync to 1. The client operating in asynchronous mode must allocate completion event object for each output buffer and pass the completion event object in the [NvEncEncodePicture\(\)](#) API. The client can create another thread and wait on the event object to be signalled by NvEncodeAPI interface on completion of the encoding process for the output frame. This should unblock the main thread from submitting work to the encoder. When the event is signalled the client can call NvEncodeAPI interfaces to copy the bitstream data using [NvEncLockBitstream\(\)](#) API. This is the preferred mode of operation.

NOTE: Asynchronous mode is not supported on Linux.

#### Synchronous mode of operation:

The client can select synchronous mode by setting NV\_ENC\_INITIALIZE\_PARAMS::enableEncodeAsync to 0. The client working in synchronous mode can work in a single threaded or multi threaded mode. The client need not allocate any event objects. The client can only lock the bitstream data after NvEncodeAPI interface has returned [NV\\_ENC\\_SUCCESS](#) from encode picture. The NvEncodeAPI interface can return [NV\\_ENC\\_ERR\\_NEED\\_MORE\\_INPUT](#) error code from [NvEncEncodePicture\(\)](#) API. The client must not lock the output buffer in such case but should send the next frame for encoding. The client must keep on calling [NvEncEncodePicture\(\)](#) API until it returns [NV\\_ENC\\_SUCCESS](#).

The client must always lock the bitstream data in order in which it has submitted. This is true for both asynchronous and synchronous mode.

#### Picture type decision:

If the client is taking the picture type decision and it must disable the picture type decision module in NvEncodeAPI by setting NV\_ENC\_INITIALIZE\_PARAMS::enablePTD to 0. In this case the client is required to send the picture in encoding order to NvEncodeAPI by doing the re-ordering for B frames.

If the client doesn't want to take the picture type decision it can enable picture type decision module in the NvEncodeAPI interface by setting NV\_ENC\_INITIALIZE\_PARAMS::enablePTD to 1 and send the input pictures in display order.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *createEncodeParams* Refer `_NV_ENC_INITIALIZE_PARAMS` for details.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.22 NVENCSTATUS NVENCAPI NvEncInvalidateRefFrames (void \* *encoder*, uint64\_t *invalidRefFrameTimeStamp*)

Invalidates reference frame based on the time stamp provided by the client. The encoder marks any reference frames or any frames which have been reconstructed using the corrupt frame as invalid for motion estimation and uses older reference frames for motion estimation. The encoder forces the current frame to be encoded as an intra frame if no reference frames are left after invalidation process. This is useful for low latency application for error resiliency. The client is recommended to set `NV_ENC_CONFIG_H264::maxNumRefFrames` to a large value so that encoder can keep a backup of older reference frames in the DPB and can use them for motion estimation when the newer reference frames have been invalidated. This API can be called multiple times.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *invalidRefFrameTimeStamp* Timestamp of the invalid reference frames which needs to be invalidated.

**Returns:**

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.23 NVENCSTATUS NVENCAPI NvEncLockBitstream (void \* *encoder*, NV\_ENC\_LOCK\_BITSTREAM \* *lockBitstreamBufferParams*)

This function is used to lock the bitstream buffer to read the encoded data. The client can only access the encoded data by calling this function. The pointer to client accessible encoded data is returned in the `NV_ENC_LOCK_BITSTREAM::bitstreamBufferPtr` field. The size of the encoded data in the output buffer is returned in the `NV_ENC_LOCK_BITSTREAM::bitstreamSizeInBytes`. The NvEncodeAPI interface also returns the output picture type and picture structure of the encoded frame in `NV_ENC_LOCK_BITSTREAM::pictureType` and `NV_ENC_LOCK_BITSTREAM::pictureStruct` fields respectively. If the client has set `NV_ENC_LOCK_BITSTREAM::doNotWait` to

1, the function might return `NV_ENC_ERR_LOCK_BUSY` if client is operating in synchronous mode. This is not a fatal failure if `NV_ENC_LOCK_BITSTREAM::doNotWait` is set to 1. In the above case the client can retry the function after few milliseconds.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *lockBitstreamBufferParams* Pointer to the `_NV_ENC_LOCK_BITSTREAM` structure.

**Returns:**

`NV_ENC_SUCCESS`  
`NV_ENC_ERR_INVALID_PTR`  
`NV_ENC_ERR_INVALID_ENCODERDEVICE`  
`NV_ENC_ERR_DEVICE_NOT_EXIST`  
`NV_ENC_ERR_UNSUPPORTED_PARAM`  
`NV_ENC_ERR_OUT_OF_MEMORY`  
`NV_ENC_ERR_INVALID_PARAM`  
`NV_ENC_ERR_INVALID_VERSION`  
`NV_ENC_ERR_LOCK_BUSY`  
`NV_ENC_ERR_ENCODER_NOT_INITIALIZED`  
`NV_ENC_ERR_GENERIC`

**4.2.1.24 NVENCSTATUS NVENCAPI NvEncLockInputBuffer (void \* encoder, NV\_ENC\_LOCK\_INPUT\_BUFFER \* lockInputBufferParams)**

This function is used to lock the input buffer to load the uncompressed YUV pixel data into input buffer memory. The client must pass the `NV_ENC_INPUT_PTR` it had previously allocated using `NvEncCreateInputBuffer()` in the `NV_ENC_LOCK_INPUT_BUFFER::inputBuffer` field. The NvEncodeAPI interface returns pointer to client accessible input buffer memory in `NV_ENC_LOCK_INPUT_BUFFER::bufferDataPtr` field.

**Parameters:**

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *lockInputBufferParams* Pointer to the `_NV_ENC_LOCK_INPUT_BUFFER` structure

**Returns:**

`NV_ENC_SUCCESS`  
`NV_ENC_ERR_INVALID_PTR`  
`NV_ENC_ERR_INVALID_ENCODERDEVICE`  
`NV_ENC_ERR_DEVICE_NOT_EXIST`  
`NV_ENC_ERR_UNSUPPORTED_PARAM`  
`NV_ENC_ERR_OUT_OF_MEMORY`  
`NV_ENC_ERR_INVALID_PARAM`  
`NV_ENC_ERR_INVALID_VERSION`  
`NV_ENC_ERR_LOCK_BUSY`  
`NV_ENC_ERR_ENCODER_NOT_INITIALIZED`  
`NV_ENC_ERR_GENERIC`



#### 4.2.1.25 NVENCSTATUS NVENCAPI NvEncMapInputResource (void \* *encoder*, NV\_ENC\_MAP\_INPUT\_RESOURCE \* *mapInputResParams*)

Maps an externally allocated input resource [using and returns a NV\_ENC\_INPUT\_PTR which can be used for encoding in the [NvEncEncodePicture\(\)](#) function. The mapped resource is returned in the field NV\_ENC\_MAP\_INPUT\_RESOURCE::outputResourcePtr. The NvEncodeAPI interface also returns the buffer format of the mapped resource in the field NV\_ENC\_MAP\_INPUT\_RESOURCE::outbufferFmt. This function provides synchronization guarantee that any direct3d or cuda work submitted on the input buffer is completed before the buffer is used for encoding. The client should not access any input buffer while they are mapped by the encoder.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *mapInputResParams* Pointer to the [\\_NV\\_ENC\\_MAP\\_INPUT\\_RESOURCE](#) structure.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_RESOURCE\\_NOT\\_REGISTERED](#)  
[NV\\_ENC\\_ERR\\_MAP\\_FAILED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.26 NVENCSTATUS NVENCAPI NvEncodeAPICreateInstance (NV\_ENCODE\_API\_FUNCTION\_LIST \* *functionList*)

Entry Point to the NvEncodeAPI interface.

Creates an instance of the NvEncodeAPI interface, and populates the pFunctionList with function pointers to the API routines implemented by the NvEncodeAPI interface.

##### Parameters:

- *functionList*

##### Returns:

[NV\\_ENC\\_SUCCESS](#) [NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)

#### 4.2.1.27 NVENCSTATUS NVENCAPI NvEncOpenEncodeSession (void \* *device*, uint32\_t *deviceType*, void \*\* *encoder*)

Deprecated.

##### Returns:

[NV\\_ENC\\_ERR\\_INVALID\\_CALL](#)

#### 4.2.1.28 NVENCSTATUS NVENCAPI NvEncOpenEncodeSessionEx (NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS \* *openSessionExParams*, void \*\**encoder*)

Opens an encoding session and returns a pointer to the encoder interface in the *\*\*encoder* parameter. The client should start encoding process by calling this API first. The client must pass a pointer to IDirect3DDevice9/CUDA interface in the *\*device* parameter. If the creation of encoder session fails, the client must call [NvEncDestroyEncoder](#) API before exiting.

##### Parameters:

- ← *openSessionExParams* Pointer to a [NV\\_ENC\\_OPEN\\_ENCODE\\_SESSION\\_EX\\_PARAMS](#) structure.
- *encoder* Encode Session pointer to the NvEncodeAPI interface.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_NO\\_ENCODE\\_DEVICE](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_DEVICE](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_DEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.29 NVENCSTATUS NVENCAPI NvEncReconfigureEncoder (void \* *encoder*, NV\_ENC\_RECONFIGURE\_PARAMS \* *reInitEncodeParams*)

Reconfigure an existing encoding session. The client should call this API to change/reconfigure the parameter passed during NvEncInitializeEncoder API call. Currently Reconfiguration of following are not supported. Change in GOP structure. Change in sync-Async mode. Change in MaxWidth & MaxHeight. Change in PTDmode.

Resolution change is possible only if maxEncodeWidth & maxEncodeHeight of NV\_ENC\_INITIALIZE\_PARAMS is set while creating encoder session.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *reInitEncodeParams* Pointer to a [NV\\_ENC\\_RECONFIGURE\\_PARAMS](#) structure.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_NO\\_ENCODE\\_DEVICE](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_DEVICE](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_DEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.30 NVENCSTATUS NVENCAPI NvEncRegisterAsyncEvent (void \* *encoder*, NV\_ENC\_EVENT\_PARAMS \* *eventParams*)

This function is used to register the completion event with NvEncodeAPI interface. The event is required when the client has configured the encoder to work in asynchronous mode. In this mode the client needs to send a completion event with every output buffer. The NvEncodeAPI interface will signal the completion of the encoding process using this event. Only after the event is signalled the client can get the encoded data using [NvEncLockBitstream\(\)](#) function.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *eventParams* Pointer to the `_NV_ENC_EVENT_PARAMS` structure.

##### Returns:

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
 NV\_ENC\_ERR\_GENERIC

#### 4.2.1.31 NVENCSTATUS NVENCAPI NvEncRegisterResource (void \* *encoder*, NV\_ENC\_REGISTER\_RESOURCE \* *registerResParams*)

Registers a resource with the Nvidia Video Encoder Interface for book keeping. The client is expected to pass the registered resource handle as well, while calling [NvEncMapInputResource](#) API. This API is not implemented for the DirectX Interface. DirectX based clients need not change their implementation.

##### Parameters:

- ← *encoder* Pointer to the NVEncodeAPI interface.
- ← *registerResParams* Pointer to a `_NV_ENC_REGISTER_RESOURCE` structure

##### Returns:

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
 NV\_ENC\_ERR\_RESOURCE\_REGISTER\_FAILED  
 NV\_ENC\_ERR\_GENERIC  
 NV\_ENC\_ERR\_UNIMPLEMENTED

#### 4.2.1.32 NVENCSTATUS NVENCAPI NvEncRunMotionEstimationOnly (void \* *encoder*, NV\_ENC\_MEONLY\_PARAMS \* *MEOnlyParams*)

This function is used to submit the input frame and reference frame for motion estimation. The ME parameters are passed using \*MEOnlyParams which is a pointer to NV\_ENC\_MEONLY\_PARAMS structure. The output motion vector data will be returned to the buffer NV\_ENC\_MEONLY\_PARAMS::outputMV.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *MEOnlyParams* Pointer to the [\\_NV\\_ENC\\_MEONLY\\_PARAMS](#) structure.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_NEED\\_MORE\\_INPUT](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.33 NVENCSTATUS NVENCAPI NvEncUnlockBitstream (void \* *encoder*, NV\_ENC\_OUTPUT\_PTR *bitstreamBuffer*)

This function is used to unlock the output bitstream buffer after the client has read the encoded data from output buffer. The client must call this function to unlock the output buffer which it has previously locked using [NvEncLockBitstream\(\)](#) function. Using a locked bitstream buffer in [NvEncEncodePicture\(\)](#) API will cause the function to fail.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ↔ *bitstreamBuffer* bitstream buffer pointer being unlocked

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.34 NVENCSTATUS NVENCAPI NvEncUnlockInputBuffer (void \* *encoder*, NV\_ENC\_INPUT\_PTR *inputBuffer*)

This function is used to unlock the input buffer memory previously locked for uploading YUV pixel data. The input buffer must be unlocked before being used again for encoding, otherwise NvEncodeAPI will fail the [NvEncEncodePicture\(\)](#)

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *inputBuffer* Pointer to the input buffer that is being unlocked.

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.35 NVENCSTATUS NVENCAPI NvEncUnmapInputResource (void \* *encoder*, NV\_ENC\_INPUT\_PTR *mappedInputBuffer*)

UnMaps an input buffer which was previously mapped using [NvEncMapInputResource\(\)](#) API. The mapping created using [NvEncMapInputResource\(\)](#) should be invalidated using this API before the external resource is destroyed by the client. The client must unmap the buffer after [NvEncLockBitstream\(\)](#) API returns successfully for encode work submitted using the mapped input buffer.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *mappedInputBuffer* Pointer to the NV\_ENC\_INPUT\_PTR

##### Returns:

[NV\\_ENC\\_SUCCESS](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PTR](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_ENCODERDEVICE](#)  
[NV\\_ENC\\_ERR\\_DEVICE\\_NOT\\_EXIST](#)  
[NV\\_ENC\\_ERR\\_UNSUPPORTED\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_OUT\\_OF\\_MEMORY](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_VERSION](#)  
[NV\\_ENC\\_ERR\\_INVALID\\_PARAM](#)  
[NV\\_ENC\\_ERR\\_ENCODER\\_NOT\\_INITIALIZED](#)  
[NV\\_ENC\\_ERR\\_RESOURCE\\_NOT\\_REGISTERED](#)  
[NV\\_ENC\\_ERR\\_RESOURCE\\_NOT\\_MAPPED](#)  
[NV\\_ENC\\_ERR\\_GENERIC](#)

#### 4.2.1.36 NVENCSTATUS NVENCAPI NvEncUnregisterAsyncEvent (void \* *encoder*, NV\_ENC\_EVENT\_PARAMS \* *eventParams*)

This function is used to unregister completion event which has been previously registered using [NvEncRegisterAsyncEvent\(\)](#) function. The client must unregister all events before destroying the encoder using [NvEncDestroyEncoder\(\)](#) function.

##### Parameters:

- ← *encoder* Pointer to the NvEncodeAPI interface.
- ← *eventParams* Pointer to the \_NV\_ENC\_EVENT\_PARAMS structure.

##### Returns:

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
 NV\_ENC\_ERR\_GENERIC

#### 4.2.1.37 NVENCSTATUS NVENCAPI NvEncUnregisterResource (void \* *encoder*, NV\_ENC\_REGISTERED\_PTR *registeredResource*)

Unregisters a resource previously registered with the Nvidia Video Encoder Interface. The client is expected to unregister any resource that it has registered with the Nvidia Video Encoder Interface before destroying the resource. This API is not implemented for the DirectX Interface. DirectX based clients need not change their implementation.

##### Parameters:

- ← *encoder* Pointer to the NVEncodeAPI interface.
- ← *registeredResource* The registered resource pointer that was returned in [NvEncRegisterResource](#).

##### Returns:

NV\_ENC\_SUCCESS  
 NV\_ENC\_ERR\_INVALID\_PTR  
 NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
 NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
 NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
 NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
 NV\_ENC\_ERR\_INVALID\_VERSION  
 NV\_ENC\_ERR\_INVALID\_PARAM  
 NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
 NV\_ENC\_ERR\_RESOURCE\_NOT\_REGISTERED  
 NV\_ENC\_ERR\_GENERIC  
 NV\_ENC\_ERR\_UNIMPLEMENTED

## Chapter 5

# Data Structure Documentation

### 5.1 `_NV_ENC_CODEC_CONFIG` Struct Reference

```
#include <nvEncodeAPI.h>
```

#### 5.1.1 Detailed Description

Codec-specific encoder configuration parameters to be set during initialization.

## 5.2 `_NV_ENC_CONFIG` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.2.1 Detailed Description

Encoder configuration parameters to be set during initialization.



## 5.3 `_NV_ENC_CONFIG_H264` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.3.1 Detailed Description

H264 encoder configuration parameters

## 5.4 `_NV_ENC_CONFIG_H264_VUI_PARAMETERS` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.4.1 Detailed Description

H264 Video Usability Info parameters

## 5.5 `_NV_ENC_CONFIG_HEVC` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.5.1 Detailed Description

HEVC encoder configuration parameters to be set during initialization.

## 5.6 `_NV_ENC_INITIALIZE_PARAMS` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.6.1 Detailed Description

Encode Session Initialization parameters.

## 5.7 `_NV_ENC_LOCK_BITSTREAM` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.7.1 Detailed Description

Bitstream buffer lock parameters.

## 5.8 `_NV_ENC_LOCK_INPUT_BUFFER` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.8.1 Detailed Description

Uncompressed Input Buffer lock parameters.

## 5.9 `_NV_ENC_MAP_INPUT_RESOURCE` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.9.1 Detailed Description

Map an input resource to a Nvidia Encoder Input Buffer

## 5.10 `_NV_ENC_MEONLY_PARAMS` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.10.1 Detailed Description

MEOnly parameters that need to be sent on a per motion estimation basis.



## 5.11 \_NV\_ENC\_PIC\_PARAMS Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.11.1 Detailed Description

Encoding parameters that need to be sent on a per frame basis.

## 5.12 `_NV_ENC_PIC_PARAMS_H264` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.12.1 Detailed Description

H264 specific enc pic params. sent on a per frame basis.

## 5.13 \_NV\_ENC\_PIC\_PARAMS\_HEVC Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.13.1 Detailed Description

HEVC specific enc pic params. sent on a per frame basis.

## 5.14 `_NV_ENC_PRESET_CONFIG` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.14.1 Detailed Description

Encoder preset config

## 5.15 `_NV_ENC_RECONFIGURE_PARAMS` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.15.1 Detailed Description

Encode Session Reconfigured parameters.

## 5.16 `_NV_ENC_REGISTER_RESOURCE` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.16.1 Detailed Description

Register a resource for future use with the Nvidia Video Encoder Interface.

## 5.17 \_NV\_ENC\_SEI\_PAYLOAD Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.17.1 Detailed Description

User SEI message

## 5.18 `_NV_ENC_SEQUENCE_PARAM_PAYLOAD` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.18.1 Detailed Description

Sequence and picture parameters payload.



## 5.19 \_NV\_ENC\_STAT Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.19.1 Detailed Description

Encode Stats structure.

## 5.20 `_NVENC_EXTERNAL_ME_HINT` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.20.1 Detailed Description

External Motion Vector hint structure.

## 5.21 \_NVENC\_EXTERNAL\_ME\_HINT\_COUNTS\_PER\_BLOCKTYPE Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.21.1 Detailed Description

External motion vector hint counts per block type.

## 5.22 `_NVENC_RECT` Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.22.1 Detailed Description

Defines a Rectangle. Used in `NV_ENC_PREPROCESS_FRAME`.

## 5.23 GUID Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [uint32\\_t Data1](#)
- [uint16\\_t Data2](#)
- [uint16\\_t Data3](#)
- [uint8\\_t Data4](#) [8]

### 5.23.1 Detailed Description

Abstracts the [GUID](#) structure for non-windows platforms.

### 5.23.2 Field Documentation

#### 5.23.2.1 `uint32_t GUID::Data1`

[in]: Specifies the first 8 hexadecimal digits of the [GUID](#).

#### 5.23.2.2 `uint16_t GUID::Data2`

[in]: Specifies the first group of 4 hexadecimal digits.

#### 5.23.2.3 `uint16_t GUID::Data3`

[in]: Specifies the second group of 4 hexadecimal digits.

#### 5.23.2.4 `uint8_t GUID::Data4[8]`

[in]: Array of 8 bytes. The first 2 bytes contain the third group of 4 hexadecimal digits. The remaining 6 bytes contain the final 12 hexadecimal digits.

## 5.24 NV\_ENC\_CAPS\_PARAM Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [uint32\\_t version](#)
- [NV\\_ENC\\_CAPS capsToQuery](#)
- [uint32\\_t reserved](#) [62]

### 5.24.1 Detailed Description

Input struct for querying Encoding capabilities.

### 5.24.2 Field Documentation

#### 5.24.2.1 NV\_ENC\_CAPS NV\_ENC\_CAPS\_PARAM::capsToQuery

[in]: Specifies the encode capability to be queried. Client should pass a member for [NV\\_ENC\\_CAPS](#) enum.

#### 5.24.2.2 [uint32\\_t](#) NV\_ENC\_CAPS\_PARAM::reserved[62]

[in]: Reserved and must be set to 0

#### 5.24.2.3 [uint32\\_t](#) NV\_ENC\_CAPS\_PARAM::version

[in]: Struct version. Must be set to [NV\\_ENC\\_CAPS\\_PARAM\\_VER](#)

## 5.25 NV\_ENC\_CODEC\_PIC\_PARAMS Union Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- NV\_ENC\_CODEC\_PIC\_PARAMS\_H264 [h264PicParams](#)
- NV\_ENC\_CODEC\_PIC\_PARAMS\_HEVC [hevcPicParams](#)
- uint32\_t [reserved](#) [256]

### 5.25.1 Detailed Description

Codec specific per-picture encoding parameters.

### 5.25.2 Field Documentation

#### 5.25.2.1 NV\_ENC\_CODEC\_PIC\_PARAMS\_H264 NV\_ENC\_CODEC\_PIC\_PARAMS::h264PicParams

[in]: H264 encode picture params.

#### 5.25.2.2 NV\_ENC\_CODEC\_PIC\_PARAMS\_HEVC NV\_ENC\_CODEC\_PIC\_PARAMS::hevcPicParams

[in]: HEVC encode picture params. Currently unsupported and must not to be used.

#### 5.25.2.3 uint32\_t NV\_ENC\_CODEC\_PIC\_PARAMS::reserved[256]

[in]: Reserved and must be set to 0.

## 5.26 NV\_ENC\_CREATE\_BITSTREAM\_BUFFER Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- `uint32_t version`
- `uint32_t size`
- `NV_ENC_MEMORY_HEAP memoryHeap`
- `uint32_t reserved`
- `NV_ENC_OUTPUT_PTR bitstreamBuffer`
- `void * bitstreamBufferPtr`
- `uint32_t reserved1 [58]`
- `void * reserved2 [64]`

### 5.26.1 Detailed Description

Creation parameters for output bitstream buffer.

### 5.26.2 Field Documentation

#### 5.26.2.1 NV\_ENC\_OUTPUT\_PTR NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::bitstreamBuffer

[out]: Pointer to the output bitstream buffer

#### 5.26.2.2 void\* NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::bitstreamBufferPtr

[out]: Reserved and should not be used

#### 5.26.2.3 NV\_ENC\_MEMORY\_HEAP NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::memoryHeap

[in]: Output buffer memory heap

#### 5.26.2.4 uint32\_t NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::reserved

[in]: Reserved and must be set to 0

#### 5.26.2.5 uint32\_t NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::reserved1[58]

[in]: Reserved and should be set to 0

#### 5.26.2.6 void\* NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::reserved2[64]

[in]: Reserved and should be set to NULL



**5.26.2.7 uint32\_t NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::size**

[in]: Size of the bitstream buffer to be created

**5.26.2.8 uint32\_t NV\_ENC\_CREATE\_BITSTREAM\_BUFFER::version**

[in]: Struct version. Must be set to [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER\\_VER](#)

## 5.27 NV\_ENC\_CREATE\_INPUT\_BUFFER Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- uint32\_t [version](#)
- uint32\_t [width](#)
- uint32\_t [height](#)
- NV\_ENC\_MEMORY\_HEAP [memoryHeap](#)
- NV\_ENC\_BUFFER\_FORMAT [bufferFmt](#)
- uint32\_t [reserved](#)
- NV\_ENC\_INPUT\_PTR [inputBuffer](#)
- void \* [pSysMemBuffer](#)
- uint32\_t [reserved1](#) [57]
- void \* [reserved2](#) [63]

### 5.27.1 Detailed Description

Creation parameters for input buffer.

### 5.27.2 Field Documentation

#### 5.27.2.1 NV\_ENC\_BUFFER\_FORMAT NV\_ENC\_CREATE\_INPUT\_BUFFER::bufferFmt

[in]: Input buffer format

#### 5.27.2.2 uint32\_t NV\_ENC\_CREATE\_INPUT\_BUFFER::height

[in]: Input buffer width

#### 5.27.2.3 NV\_ENC\_INPUT\_PTR NV\_ENC\_CREATE\_INPUT\_BUFFER::inputBuffer

[out]: Pointer to input buffer

#### 5.27.2.4 NV\_ENC\_MEMORY\_HEAP NV\_ENC\_CREATE\_INPUT\_BUFFER::memoryHeap

[in]: Input buffer memory heap

#### 5.27.2.5 void\* NV\_ENC\_CREATE\_INPUT\_BUFFER::pSysMemBuffer

[in]: Pointer to existing system buffer

#### 5.27.2.6 uint32\_t NV\_ENC\_CREATE\_INPUT\_BUFFER::reserved

[in]: Reserved and must be set to 0

**5.27.2.7 uint32\_t NV\_ENC\_CREATE\_INPUT\_BUFFER::reserved1[57]**

[in]: Reserved and must be set to 0

**5.27.2.8 void\* NV\_ENC\_CREATE\_INPUT\_BUFFER::reserved2[63]**

[in]: Reserved and must be set to NULL

**5.27.2.9 uint32\_t NV\_ENC\_CREATE\_INPUT\_BUFFER::version**

[in]: Struct version. Must be set to [NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER\\_VER](#)

**5.27.2.10 uint32\_t NV\_ENC\_CREATE\_INPUT\_BUFFER::width**

[in]: Input buffer width

## 5.28 NV\_ENC\_CREATE\_MV\_BUFFER Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- `uint32_t` [version](#)
- `NV_ENC_OUTPUT_PTR` [MVBuffer](#)
- `uint32_t` [reserved1](#) [255]
- `void *` [reserved2](#) [63]

### 5.28.1 Detailed Description

Creation parameters for output motion vector buffer for ME only mode.

### 5.28.2 Field Documentation

#### 5.28.2.1 NV\_ENC\_OUTPUT\_PTR NV\_ENC\_CREATE\_MV\_BUFFER::MVBuffer

[out]: Pointer to the output MV buffer

#### 5.28.2.2 uint32\_t NV\_ENC\_CREATE\_MV\_BUFFER::reserved1[255]

[in]: Reserved and should be set to 0

#### 5.28.2.3 void\* NV\_ENC\_CREATE\_MV\_BUFFER::reserved2[63]

[in]: Reserved and should be set to NULL

#### 5.28.2.4 uint32\_t NV\_ENC\_CREATE\_MV\_BUFFER::version

[in]: Struct version. Must be set to NV\_ENC\_CREATE\_MV\_BUFFER\_VER

## 5.29 NV\_ENC\_EVENT\_PARAMS Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [uint32\\_t version](#)
- [uint32\\_t reserved](#)
- [void \\* completionEvent](#)
- [uint32\\_t reserved1 \[253\]](#)
- [void \\* reserved2 \[64\]](#)

### 5.29.1 Detailed Description

Event registration/unregistration parameters.

### 5.29.2 Field Documentation

#### 5.29.2.1 void\* NV\_ENC\_EVENT\_PARAMS::completionEvent

[in]: Handle to event to be registered/unregistered with the NvEncodeAPI interface.

#### 5.29.2.2 uint32\_t NV\_ENC\_EVENT\_PARAMS::reserved

[in]: Reserved and must be set to 0

#### 5.29.2.3 uint32\_t NV\_ENC\_EVENT\_PARAMS::reserved1[253]

[in]: Reserved and must be set to 0

#### 5.29.2.4 void\* NV\_ENC\_EVENT\_PARAMS::reserved2[64]

[in]: Reserved and must be set to NULL

#### 5.29.2.5 uint32\_t NV\_ENC\_EVENT\_PARAMS::version

[in]: Struct version. Must be set to [NV\\_ENC\\_EVENT\\_PARAMS\\_VER](#).

## 5.30 NV\_ENC\_H264\_MV\_DATA Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [NV\\_ENC\\_MVECTOR MV](#) [4]
- [uint8\\_t mb\\_type](#)
- [uint8\\_t partitionType](#)
- [uint16\\_t reserved](#)

### 5.30.1 Detailed Description

Description of MV structure per macroblock for ME only mode.

### 5.30.2 Field Documentation

#### 5.30.2.1 `uint8_t NV_ENC_H264_MV_DATA::mb_type`

0 (I), 1 (P), 2 (IPCM), 3 (B)

#### 5.30.2.2 `NV_ENC_MVECTOR NV_ENC_H264_MV_DATA::MV[4]`

up to 4 vectors for 8x8 partition

#### 5.30.2.3 `uint8_t NV_ENC_H264_MV_DATA::partitionType`

Specifies the block partition type. 0:16x16, 1:8x8, 2:16x8, 3:8x16

#### 5.30.2.4 `uint16_t NV_ENC_H264_MV_DATA::reserved`

reserved padding for alignment

## 5.31 NV\_ENC\_MVECTOR Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [int16\\_t mvx](#)
- [int16\\_t mvy](#)

### 5.31.1 Detailed Description

Structs needed for ME only mode.

### 5.31.2 Field Documentation

#### 5.31.2.1 int16\_t NV\_ENC\_MVECTOR::mvx

the x component of MV in qpel units

#### 5.31.2.2 int16\_t NV\_ENC\_MVECTOR::mvy

the y component of MV in qpel units

## 5.32 NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [uint32\\_t version](#)
- [NV\\_ENC\\_DEVICE\\_TYPE deviceType](#)
- void \* [device](#)
- void \* [reserved](#)
- [uint32\\_t apiVersion](#)
- [uint32\\_t reserved1 \[253\]](#)
- void \* [reserved2 \[64\]](#)

### 5.32.1 Detailed Description

Encoder Session Creation parameters

### 5.32.2 Field Documentation

#### 5.32.2.1 `uint32_t NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::apiVersion`

[in]: API version. Should be set to NVENCAPI\_VERSION.

#### 5.32.2.2 `void* NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::device`

[in]: Pointer to client device.

#### 5.32.2.3 `NV_ENC_DEVICE_TYPE NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::deviceType`

[in]: Specified the device Type

#### 5.32.2.4 `void* NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved`

[in]: Reserved and must be set to 0.

#### 5.32.2.5 `uint32_t NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved1[253]`

[in]: Reserved and must be set to 0

#### 5.32.2.6 `void* NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved2[64]`

[in]: Reserved and must be set to NULL



**5.32.2.7 uint32\_t NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS::version**

[in]: Struct version. Must be set to [NV\\_ENC\\_OPEN\\_ENCODE\\_SESSION\\_EX\\_PARAMS\\_VER](#).

## 5.33 NV\_ENC\_QP Struct Reference

```
#include <nvEncodeAPI.h>
```

### 5.33.1 Detailed Description

QP value for frames

## 5.34 NV\_ENC\_RC\_PARAMS Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- [NV\\_ENC\\_PARAMS\\_RC\\_MODE](#) rateControlMode
- [NV\\_ENC\\_QP](#) constQP
- [uint32\\_t](#) averageBitRate
- [uint32\\_t](#) maxBitRate
- [uint32\\_t](#) vbvBufferSize
- [uint32\\_t](#) vbvInitialDelay
- [uint32\\_t](#) enableMinQP:1
- [uint32\\_t](#) enableMaxQP:1
- [uint32\\_t](#) enableInitialRCQP:1
- [uint32\\_t](#) enableAQ:1
- [uint32\\_t](#) enableExtQPDeltaMap:1
- [uint32\\_t](#) reservedBitFields:27
- [NV\\_ENC\\_QP](#) minQP
- [NV\\_ENC\\_QP](#) maxQP
- [NV\\_ENC\\_QP](#) initialRCQP
- [uint32\\_t](#) temporallayerIdxMask
- [uint8\\_t](#) temporalLayerQP [8]

### 5.34.1 Detailed Description

Rate Control Configuration Paramters

### 5.34.2 Field Documentation

#### 5.34.2.1 [uint32\\_t](#) NV\_ENC\_RC\_PARAMS::averageBitRate

[in]: Specifies the average bitrate(in bits/sec) used for encoding.

#### 5.34.2.2 [NV\\_ENC\\_QP](#) NV\_ENC\_RC\_PARAMS::constQP

[in]: Specifies the initial QP to be used for encoding, these values would be used for all frames if in Constant QP mode.

#### 5.34.2.3 [uint32\\_t](#) NV\_ENC\_RC\_PARAMS::enableAQ

[in]: Set this to 1 to enable adaptive quantization.

#### 5.34.2.4 [uint32\\_t](#) NV\_ENC\_RC\_PARAMS::enableExtQPDeltaMap

[in]: Set this to 1 to enable additional QP modifier for each MB supplied by client though signed byte array pointed to by NV\_ENC\_PIC\_PARAMS::qpDeltaMap

**5.34.2.5 uint32\_t NV\_ENC\_RC\_PARAMS::enableInitialRCQP**

[in]: Set this to 1 if user supplied initial QP is used for rate control.

**5.34.2.6 uint32\_t NV\_ENC\_RC\_PARAMS::enableMaxQP**

[in]: Set this to 1 if maximum QP used for rate control.

**5.34.2.7 uint32\_t NV\_ENC\_RC\_PARAMS::enableMinQP**

[in]: Set this to 1 if minimum QP used for rate control.

**5.34.2.8 NV\_ENC\_QP NV\_ENC\_RC\_PARAMS::initialRCQP**

[in]: Specifies the initial QP used for rate control. Client must set NV\_ENC\_CONFIG::enableInitialRCQP to 1.

**5.34.2.9 uint32\_t NV\_ENC\_RC\_PARAMS::maxBitRate**

[in]: Specifies the maximum bitrate for the encoded output. This is used for VBR and ignored for CBR mode.

**5.34.2.10 NV\_ENC\_QP NV\_ENC\_RC\_PARAMS::maxQP**

[in]: Specifies the maximum QP used for rate control. Client must set NV\_ENC\_CONFIG::enableMaxQP to 1.

**5.34.2.11 NV\_ENC\_QP NV\_ENC\_RC\_PARAMS::minQP**

[in]: Specifies the minimum QP used for rate control. Client must set NV\_ENC\_CONFIG::enableMinQP to 1.

**5.34.2.12 NV\_ENC\_PARAMS\_RC\_MODE NV\_ENC\_RC\_PARAMS::rateControlMode**

[in]: Specifies the rate control mode. Check support for various rate control modes using [NV\\_ENC\\_CAPS\\_SUPPORTED\\_RATECONTROL\\_MODES](#) caps.

**5.34.2.13 uint32\_t NV\_ENC\_RC\_PARAMS::reservedBitFields**

[in]: Reserved bitfields and must be set to 0

**5.34.2.14 uint32\_t NV\_ENC\_RC\_PARAMS::temporalLayerIdxMask**

[in]: Specifies the temporal layers (as a bitmask) whose QPs have changed. Valid max bitmask is  $[2^{\text{NV\_ENC\_CAPS\_NUM\_MAX\_TEMPORAL\_LAYERS}} - 1]$

**5.34.2.15 uint8\_t NV\_ENC\_RC\_PARAMS::temporalLayerQP[8]**

[in]: Specifies the temporal layer QPs used for rate control. Temporal layer index is used as the array index

**5.34.2.16 uint32\_t NV\_ENC\_RC\_PARAMS::vbvBufferSize**

[in]: Specifies the VBV(HRD) buffer size. in bits. Set 0 to use the default VBV buffer size.

**5.34.2.17 uint32\_t NV\_ENC\_RC\_PARAMS::vbvInitialDelay**

[in]: Specifies the VBV(HRD) initial delay in bits. Set 0 to use the default VBV initial delay .

## 5.35 NV\_ENCODE\_API\_FUNCTION\_LIST Struct Reference

```
#include <nvEncodeAPI.h>
```

### Data Fields

- uint32\_t [version](#)
- uint32\_t [reserved](#)
- PNVENCOOPENENCODESESSION [nvEncOpenEncodeSession](#)
- PNVENCGETENCODEGUIDCOUNT [nvEncGetEncodeGUIDCount](#)
- PNVENCGETENCODEPRESETCOUNT [nvEncGetEncodeProfileGUIDCount](#)
- PNVENCGETENCODEPRESETGUIDS [nvEncGetEncodeProfileGUIDs](#)
- PNVENCGETENCODEGUIDS [nvEncGetEncodeGUIDs](#)
- PNVENCGETINPUTFORMATCOUNT [nvEncGetInputFormatCount](#)
- PNVENCGETINPUTFORMATS [nvEncGetInputFormats](#)
- PNVENCGETENCODECAPS [nvEncGetEncodeCaps](#)
- PNVENCGETENCODEPRESETCOUNT [nvEncGetEncodePresetCount](#)
- PNVENCGETENCODEPRESETGUIDS [nvEncGetEncodePresetGUIDs](#)
- PNVENCGETENCODEPRESETCONFIG [nvEncGetEncodePresetConfig](#)
- PNVENCINITIALIZEENCODER [nvEncInitializeEncoder](#)
- PNVENC\_CREATEINPUTBUFFER [nvEncCreateInputBuffer](#)
- PNVENC\_DESTROYINPUTBUFFER [nvEncDestroyInputBuffer](#)
- PNVENC\_CREATEBITSTREAMBUFFER [nvEncCreateBitstreamBuffer](#)
- PNVENC\_DESTROYBITSTREAMBUFFER [nvEncDestroyBitstreamBuffer](#)
- PNVENC\_ENCODEPICTURE [nvEncEncodePicture](#)
- PNVENC\_LOCKBITSTREAM [nvEncLockBitstream](#)
- PNVENC\_UNLOCKBITSTREAM [nvEncUnlockBitstream](#)
- PNVENC\_LOCKINPUTBUFFER [nvEncLockInputBuffer](#)
- PNVENC\_UNLOCKINPUTBUFFER [nvEncUnlockInputBuffer](#)
- PNVENC\_GETENCODESTATS [nvEncGetEncodeStats](#)
- PNVENC\_GETSEQUENCEPARAMS [nvEncGetSequenceParams](#)
- PNVENC\_REGISTERASYNCEVENT [nvEncRegisterAsyncEvent](#)
- PNVENC\_UNREGISTERASYNCEVENT [nvEncUnregisterAsyncEvent](#)
- PNVENC\_MAPINPUTRESOURCE [nvEncMapInputResource](#)
- PNVENC\_UNMAPINPUTRESOURCE [nvEncUnmapInputResource](#)
- PNVENC\_DESTROYENCODER [nvEncDestroyEncoder](#)
- PNVENC\_INVALIDATEREFFRAMES [nvEncInvalidateRefFrames](#)
- PNVENC\_OPENENCODESESSIONEX [nvEncOpenEncodeSessionEx](#)
- PNVENC\_REGISTERRESOURCE [nvEncRegisterResource](#)
- PNVENC\_UNREGISTERRESOURCE [nvEncUnregisterResource](#)
- PNVENC\_RECONFIGUREENCODER [nvEncReconfigureEncoder](#)
- PNVENC\_CREATEMVBUFFER [nvEncCreateMVBuffer](#)
- PNVENC\_DESTROYMVBUFFER [nvEncDestroyMVBuffer](#)
- PNVENC\_RUNMOTIONESTIMATIONONLY [nvEncRunMotionEstimationOnly](#)
- void \* [reserved2](#) [281]

### 5.35.1 Detailed Description

[NV\\_ENCODE\\_API\\_FUNCTION\\_LIST](#)

## 5.35.2 Field Documentation

### 5.35.2.1 PNVENC\_CREATE\_BITSTREAM\_BUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncCreateBitstreamBuffer

[out]: Client should access [NvEncCreateBitstreamBuffer\(\)](#) API through this pointer.

### 5.35.2.2 PNVENC\_CREATE\_INPUT\_BUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncCreateInputBuffer

[out]: Client should access [NvEncCreateInputBuffer\(\)](#) API through this pointer.

### 5.35.2.3 PNVENC\_CREATE\_MV\_BUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncCreateMVBuffer

[out]: Client should access [NvEncCreateMVBuffer](#) API through this pointer.

### 5.35.2.4 PNVENC\_DESTROY\_BITSTREAM\_BUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncDestroyBitstreamBuffer

[out]: Client should access [NvEncDestroyBitstreamBuffer\(\)](#) API through this pointer.

### 5.35.2.5 PNVENC\_DESTROY\_ENCODER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncDestroyEncoder

[out]: Client should access [NvEncDestroyEncoder\(\)](#) API through this pointer.

### 5.35.2.6 PNVENC\_DESTROY\_INPUT\_BUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncDestroyInputBuffer

[out]: Client should access [NvEncDestroyInputBuffer\(\)](#) API through this pointer.

### 5.35.2.7 PNVENC\_DESTROY\_MV\_BUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncDestroyMVBuffer

[out]: Client should access [NvEncDestroyMVBuffer](#) API through this pointer.

### 5.35.2.8 PNVENC\_ENCODE\_PICTURE NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncEncodePicture

[out]: Client should access [NvEncEncodePicture\(\)](#) API through this pointer.

### 5.35.2.9 PNVENC\_GET\_ENCODE\_CAPS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodeCaps

[out]: Client should access [NvEncGetEncodeCaps\(\)](#) API through this pointer.

### 5.35.2.10 PNVENC\_GET\_ENCODE\_GUID\_COUNT NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodeGUIDCount

[out]: Client should access [NvEncGetEncodeGUIDCount\(\)](#) API through this pointer.

**5.35.2.11 PNVENCGETENCODEGUIDS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodeGUIDs**

[out]: Client should access [NvEncGetEncodeGUIDs\(\)](#) API through this pointer.

**5.35.2.12 PNVENCGETENCODEPRESETCONFIG NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodePresetConfig**

[out]: Client should access [NvEncGetEncodePresetConfig\(\)](#) API through this pointer.

**5.35.2.13 PNVENCGETENCODEPRESETCOUNT NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodePresetCount**

[out]: Client should access [NvEncGetEncodePresetCount\(\)](#) API through this pointer.

**5.35.2.14 PNVENCGETENCODEPRESETGUIDS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodePresetGUIDs**

[out]: Client should access [NvEncGetEncodePresetGUIDs\(\)](#) API through this pointer.

**5.35.2.15 PNVENCGETENCODEPRESETCOUNT NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodeProfileGUIDCount**

[out]: Client should access [NvEncGetEncodeProfileGUIDCount\(\)](#) API through this pointer.

**5.35.2.16 PNVENCGETENCODEPRESETGUIDS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodeProfileGUIDs**

[out]: Client should access [NvEncGetEncodeProfileGUIDs\(\)](#) API through this pointer.

**5.35.2.17 PNVENCGETENCODESTATS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetEncodeStats**

[out]: Client should access [NvEncGetEncodeStats\(\)](#) API through this pointer.

**5.35.2.18 PNVENCGETINPUTFORMATCOUNT NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetInputFormatCount**

[out]: Client should access [NvEncGetInputFormatCount\(\)](#) API through this pointer.

**5.35.2.19 PNVENCGETINPUTFORMATS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetInputFormats**

[out]: Client should access [NvEncGetInputFormats\(\)](#) API through this pointer.

**5.35.2.20 PNVENCGETSEQUENCEPARAMS NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncGetSequenceParams**

[out]: Client should access [NvEncGetSequenceParams\(\)](#) API through this pointer.



**5.35.2.21 PNVENCINITIALIZEENCODER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncInitializeEncoder**

[out]: Client should access [NvEncInitializeEncoder\(\)](#) API through this pointer.

**5.35.2.22 PNVENCINVALIDATEREFFRAMES NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncInvalidateRefFrames**

[out]: Client should access [NvEncInvalidateRefFrames\(\)](#) API through this pointer.

**5.35.2.23 PNVENCLOCKBITSTREAM NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncLockBitstream**

[out]: Client should access [NvEncLockBitstream\(\)](#) API through this pointer.

**5.35.2.24 PNVENCLOCKINPUTBUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncLockInputBuffer**

[out]: Client should access [NvEncLockInputBuffer\(\)](#) API through this pointer.

**5.35.2.25 PNVENCMAPINPUTRESOURCE NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncMapInputResource**

[out]: Client should access [NvEncMapInputResource\(\)](#) API through this pointer.

**5.35.2.26 PNVENCOPENENCODESESSION NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncOpenEncodeSession**

[out]: Client should access [NvEncOpenEncodeSession\(\)](#) API through this pointer.

**5.35.2.27 PNVENCOPENENCODESESSIONEX NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncOpenEncodeSessionEx**

[out]: Client should access [NvEncOpenEncodeSession\(\)](#) API through this pointer.

**5.35.2.28 PNVENCRECONFIGUREENCODER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncReconfigureEncoder**

[out]: Client should access [NvEncReconfigureEncoder\(\)](#) API through this pointer.

**5.35.2.29 PNVENCREGISTERASYNC EVENT NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncRegisterAsyncEvent**

[out]: Client should access [NvEncRegisterAsyncEvent\(\)](#) API through this pointer.

**5.35.2.30 PNVENCREGISTERRESOURCE NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncRegisterResource**

[out]: Client should access [NvEncRegisterResource\(\)](#) API through this pointer.

**5.35.2.31 PNVENCRUNMOTIONESTIMATIONONLY NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncRunMotionEstimationOnly**

[out]: Client should access [NvEncRunMotionEstimationOnly](#) API through this pointer.

**5.35.2.32 PNVENCUNLOCKBITSTREAM NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncUnlockBitstream**

[out]: Client should access [NvEncUnlockBitstream\(\)](#) API through this pointer.

**5.35.2.33 PNVENCUNLOCKINPUTBUFFER NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncUnlockInputBuffer**

[out]: Client should access [NvEncUnlockInputBuffer\(\)](#) API through this pointer.

**5.35.2.34 PNVENCUNMAPINPUTRESOURCE NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncUnmapInputResource**

[out]: Client should access [NvEncUnmapInputResource\(\)](#) API through this pointer.

**5.35.2.35 PNVENCUNREGISTERASYNCEVENT NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncUnregisterAsyncEvent**

[out]: Client should access [NvEncUnregisterAsyncEvent\(\)](#) API through this pointer.

**5.35.2.36 PNVENCUNREGISTERRESOURCE NV\_ENCODE\_API\_FUNCTION\_LIST::nvEncUnregisterResource**

[out]: Client should access [NvEncUnregisterResource\(\)](#) API through this pointer.

**5.35.2.37 uint32\_t NV\_ENCODE\_API\_FUNCTION\_LIST::reserved**

[in]: Reserved and should be set to 0.

**5.35.2.38 void\* NV\_ENCODE\_API\_FUNCTION\_LIST::reserved2[281]**

[in]: Reserved and must be set to NULL

**5.35.2.39 uint32\_t NV\_ENCODE\_API\_FUNCTION\_LIST::version**

[in]: Client should pass NV\_ENCODE\_API\_FUNCTION\_LIST\_VER.

# Index

- [\\_NVENC\\_EXTERNAL\\_ME\\_HINT](#), 64
- [\\_NVENC\\_EXTERNAL\\_ME\\_HINT\\_COUNTS\\_PER\\_BLOCKTYPE](#), 65
- [\\_NVENC\\_RECT](#), 66
- [\\_NV\\_ENC\\_CODEC\\_CONFIG](#), 45
- [\\_NV\\_ENC\\_CONFIG](#), 46
- [\\_NV\\_ENC\\_CONFIG\\_H264](#), 47
- [\\_NV\\_ENC\\_CONFIG\\_H264\\_VUI\\_PARAMETERS](#), 48
- [\\_NV\\_ENC\\_CONFIG\\_HEVC](#), 49
- [\\_NV\\_ENC\\_INITIALIZE\\_PARAMS](#), 50
- [\\_NV\\_ENC\\_LOCK\\_BITSTREAM](#), 51
- [\\_NV\\_ENC\\_LOCK\\_INPUT\\_BUFFER](#), 52
- [\\_NV\\_ENC\\_MAP\\_INPUT\\_RESOURCE](#), 53
- [\\_NV\\_ENC\\_MEONLY\\_PARAMS](#), 54
- [\\_NV\\_ENC\\_PIC\\_PARAMS](#), 55
- [\\_NV\\_ENC\\_PIC\\_PARAMS\\_H264](#), 56
- [\\_NV\\_ENC\\_PIC\\_PARAMS\\_HEVC](#), 57
- [\\_NV\\_ENC\\_PRESET\\_CONFIG](#), 58
- [\\_NV\\_ENC\\_RECONFIGURE\\_PARAMS](#), 59
- [\\_NV\\_ENC\\_REGISTER\\_RESOURCE](#), 60
- [\\_NV\\_ENC\\_SEI\\_PAYLOAD](#), 61
- [\\_NV\\_ENC\\_SEQUENCE\\_PARAM\\_PAYLOAD](#), 62
- [\\_NV\\_ENC\\_STAT](#), 63
- apiVersion
  - [NV\\_ENC\\_OPEN\\_ENCODE\\_SESSION\\_EX\\_PARAMS](#), 78
- averageBitRate
  - [NV\\_ENC\\_RC\\_PARAMS](#), 81
- bitstreamBuffer
  - [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER](#), 70
- bitstreamBufferPtr
  - [NV\\_ENC\\_CREATE\\_BITSTREAM\\_BUFFER](#), 70
- bufferFmt
  - [NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER](#), 72
- capsToQuery
  - [NV\\_ENC\\_CAPS\\_PARAM](#), 68
- completionEvent
  - [NV\\_ENC\\_EVENT\\_PARAMS](#), 75
- constQP
  - [NV\\_ENC\\_RC\\_PARAMS](#), 81
- Data1
  - [GUID](#), 67
- Data2
  - [GUID](#), 67
- Data3
  - [GUID](#), 67
- Data4
  - [GUID](#), 67
- device
  - [NV\\_ENC\\_OPEN\\_ENCODE\\_SESSION\\_EX\\_PARAMS](#), 78
- deviceType
  - [NV\\_ENC\\_OPEN\\_ENCODE\\_SESSION\\_EX\\_PARAMS](#), 78
- enableAQ
  - [NV\\_ENC\\_RC\\_PARAMS](#), 81
- enableExtQPDeltaMap
  - [NV\\_ENC\\_RC\\_PARAMS](#), 81
- enableInitialRCQP
  - [NV\\_ENC\\_RC\\_PARAMS](#), 81
- enableMaxQP
  - [NV\\_ENC\\_RC\\_PARAMS](#), 82
- enableMinQP
  - [NV\\_ENC\\_RC\\_PARAMS](#), 82
- ENCODE\_FUNC
  - [NvEncCreateBitstreamBuffer](#), 24
  - [NvEncCreateInputBuffer](#), 24
  - [NvEncCreateMVBuffer](#), 24
  - [NvEncDestroyBitstreamBuffer](#), 25
  - [NvEncDestroyEncoder](#), 25
  - [NvEncDestroyInputBuffer](#), 26
  - [NvEncDestroyMVBuffer](#), 26
  - [NvEncEncodePicture](#), 27
  - [NvEncGetEncodeCaps](#), 29
  - [NvEncGetEncodeGUIDCount](#), 30
  - [NvEncGetEncodeGUIDs](#), 30
  - [NvEncGetEncodePresetConfig](#), 31
  - [NvEncGetEncodePresetCount](#), 31
  - [NvEncGetEncodePresetGUIDs](#), 32
  - [NvEncGetEncodeProfileGUIDCount](#), 32
  - [NvEncGetEncodeProfileGUIDs](#), 33
  - [NvEncGetEncodeStats](#), 33
  - [NvEncGetInputFormatCount](#), 34
  - [NvEncGetInputFormats](#), 34
  - [NvEncGetSequenceParams](#), 35
  - [NvEncInitializeEncoder](#), 35

- NvEncInvalidateRefFrames, 37
- NvEncLockBitstream, 37
- NvEncLockInputBuffer, 38
- NvEncMapInputResource, 38
- NvEncodeAPICreateInstance, 39
- NvEncOpenEncodeSession, 39
- NvEncOpenEncodeSessionEx, 39
- NvEncReconfigureEncoder, 40
- NvEncRegisterAsyncEvent, 40
- NvEncRegisterResource, 41
- NvEncRunMotionEstimationOnly, 41
- NvEncUnlockBitstream, 42
- NvEncUnlockInputBuffer, 42
- NvEncUnmapInputResource, 43
- NvEncUnregisterAsyncEvent, 43
- NvEncUnregisterResource, 44
- ENCODER\_STRUCTURE
  - NV\_ENC\_BUFFER\_FORMAT\_ARGB, 12
  - NV\_ENC\_BUFFER\_FORMAT\_ARGB10, 12
  - NV\_ENC\_BUFFER\_FORMAT\_AYUV, 12
  - NV\_ENC\_BUFFER\_FORMAT\_IYUV, 12
  - NV\_ENC\_BUFFER\_FORMAT\_NV12, 12
  - NV\_ENC\_BUFFER\_FORMAT\_UNDEFINED, 12
  - NV\_ENC\_BUFFER\_FORMAT\_YUV444, 12
  - NV\_ENC\_BUFFER\_FORMAT\_YV12, 12
  - NV\_ENC\_CAPS\_ASYNC\_ENCODE\_SUPPORT, 14
  - NV\_ENC\_CAPS\_EXPOSED\_COUNT, 15
  - NV\_ENC\_CAPS\_HEIGHT\_MAX, 13
  - NV\_ENC\_CAPS\_LEVEL\_MAX, 13
  - NV\_ENC\_CAPS\_LEVEL\_MIN, 13
  - NV\_ENC\_CAPS\_MB\_NUM\_MAX, 14
  - NV\_ENC\_CAPS\_MB\_PER\_SEC\_MAX, 14
  - NV\_ENC\_CAPS\_NUM\_MAX\_BFRAMES, 12
  - NV\_ENC\_CAPS\_NUM\_MAX\_TEMPORAL\_LAYERS, 13
  - NV\_ENC\_CAPS\_PREPROC\_SUPPORT, 14
  - NV\_ENC\_CAPS\_SEPARATE\_COLOUR\_PLANE, 13
  - NV\_ENC\_CAPS\_SUPPORT\_ADAPTIVE\_TRANSFORM, 13
  - NV\_ENC\_CAPS\_SUPPORT\_BDIRECT\_MODE, 13
  - NV\_ENC\_CAPS\_SUPPORT\_CABAC, 13
  - NV\_ENC\_CAPS\_SUPPORT\_CONSTRAINED\_ENCODING, 14
  - NV\_ENC\_CAPS\_SUPPORT\_CUSTOM\_VBV\_BUF\_SIZE, 14
  - NV\_ENC\_CAPS\_SUPPORT\_DYN\_BITRATE\_CHANGE, 13
  - NV\_ENC\_CAPS\_SUPPORT\_DYN\_FORCE\_CONSTQP, 14
  - NV\_ENC\_CAPS\_SUPPORT\_DYN\_RCMODE\_CHANGE, 14
  - NV\_ENC\_CAPS\_SUPPORT\_DYN\_RES\_CHANGE, 13
  - NV\_ENC\_CAPS\_SUPPORT\_DYNAMIC\_SLICE\_MODE, 14
  - NV\_ENC\_CAPS\_SUPPORT\_FIELD\_ENCODING, 12
  - NV\_ENC\_CAPS\_SUPPORT\_FMO, 13
  - NV\_ENC\_CAPS\_SUPPORT\_HIERARCHICAL\_BFRAMES, 13
  - NV\_ENC\_CAPS\_SUPPORT\_HIERARCHICAL\_PFRAMES, 13
  - NV\_ENC\_CAPS\_SUPPORT\_INTRA\_REFRESH, 14
  - NV\_ENC\_CAPS\_SUPPORT\_LOSSLESS\_ENCODE, 15
  - NV\_ENC\_CAPS\_SUPPORT\_MEONLY\_MODE, 15
  - NV\_ENC\_CAPS\_SUPPORT\_MONOCHROME, 12
  - NV\_ENC\_CAPS\_SUPPORT\_QPELMV, 13
  - NV\_ENC\_CAPS\_SUPPORT\_REF\_PIC\_INVALIDATION, 14
  - NV\_ENC\_CAPS\_SUPPORT\_RESERVED, 13
  - NV\_ENC\_CAPS\_SUPPORT\_SAO, 15
  - NV\_ENC\_CAPS\_SUPPORT\_SUBFRAME\_READBACK, 14
  - NV\_ENC\_CAPS\_SUPPORT\_TEMPORAL\_SVC, 13
  - NV\_ENC\_CAPS\_SUPPORT\_YUV444\_ENCODE, 14
  - NV\_ENC\_CAPS\_SUPPORTED\_RATECONTROL\_MODES, 12
  - NV\_ENC\_CAPS\_WIDTH\_MAX, 13
  - NV\_ENC\_DEVICE\_TYPE\_CUDA, 15
  - NV\_ENC\_DEVICE\_TYPE\_DIRECTX, 15
  - NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST, 19
  - NV\_ENC\_ERR\_ENCODER\_BUSY, 19
  - NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED, 19
  - NV\_ENC\_ERR\_EVENT\_NOT\_REGISTERD, 19
  - NV\_ENC\_ERR\_GENERIC, 19
  - NV\_ENC\_ERR\_INCOMPATIBLE\_CLIENT\_KEY, 19
  - NV\_ENC\_ERR\_INVALID\_CALL, 19
  - NV\_ENC\_ERR\_INVALID\_DEVICE, 19
  - NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE, 19
  - NV\_ENC\_ERR\_INVALID\_EVENT, 19
  - NV\_ENC\_ERR\_INVALID\_PARAM, 19
  - NV\_ENC\_ERR\_INVALID\_PTR, 19
  - NV\_ENC\_ERR\_INVALID\_VERSION, 19
  - NV\_ENC\_ERR\_LOCK\_BUSY, 19
  - NV\_ENC\_ERR\_MAP\_FAILED, 19
  - NV\_ENC\_ERR\_NEED\_MORE\_INPUT, 19

- NV\_ENC\_ERR\_NO\_ENCODE\_DEVICE, 18
- NV\_ENC\_ERR\_NOT\_ENOUGH\_BUFFER, 19
- NV\_ENC\_ERR\_OUT\_OF\_MEMORY, 19
- NV\_ENC\_ERR\_RESOURCE\_NOT\_MAPPED, 20
- NV\_ENC\_ERR\_RESOURCE\_NOT\_REGISTERED, 20
- NV\_ENC\_ERR\_RESOURCE\_REGISTER\_FAILED, 20
- NV\_ENC\_ERR\_UNIMPLEMENTED, 19
- NV\_ENC\_ERR\_UNSUPPORTED\_DEVICE, 19
- NV\_ENC\_ERR\_UNSUPPORTED\_PARAM, 19
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_AUTOSELECT, 15
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_DISABLE, 15
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_ENABLE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_AUTOSELECT, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_DISABLE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_SPATIAL, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_TEMPORAL, 15
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_AUTOSELECT, 16
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_CABAC, 16
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_CAVLC, 16
- NV\_ENC\_H264\_FMO\_AUTOSELECT, 16
- NV\_ENC\_H264\_FMO\_DISABLE, 16
- NV\_ENC\_H264\_FMO\_ENABLE, 16
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDAARRAY, 16
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDEVICEPTR, 16
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_DIRECTX, 16
- NV\_ENC\_MEMORY\_HEAP\_AUTOSELECT, 16
- NV\_ENC\_MEMORY\_HEAP\_SYSMEM\_CACHED, 16
- NV\_ENC\_MEMORY\_HEAP\_SYSMEM\_UNCACHED, 16
- NV\_ENC\_MEMORY\_HEAP\_VID, 16
- NV\_ENC\_MV\_PRECISION\_DEFAULT, 17
- NV\_ENC\_MV\_PRECISION\_FULL\_PEL, 17
- NV\_ENC\_MV\_PRECISION\_HALF\_PEL, 17
- NV\_ENC\_MV\_PRECISION\_QUARTER\_PEL, 17
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_FIELD, 17
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_FRAME, 17
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_MBAFF, 17
- NV\_ENC\_PARAMS\_RC\_2\_PASS\_FRAMESIZE\_CAP, 17
- NV\_ENC\_PARAMS\_RC\_2\_PASS\_QUALITY, 17
- NV\_ENC\_PARAMS\_RC\_2\_PASS\_VBR, 17
- NV\_ENC\_PARAMS\_RC\_CBR, 17
- NV\_ENC\_PARAMS\_RC\_CONSTQP, 17
- NV\_ENC\_PARAMS\_RC\_VBR, 17
- NV\_ENC\_PARAMS\_RC\_VBR\_MINQP, 17
- NV\_ENC\_PIC\_FLAG\_EOS, 17
- NV\_ENC\_PIC\_FLAG\_FORCEIDR, 17
- NV\_ENC\_PIC\_FLAG\_FORCEINTRA, 17
- NV\_ENC\_PIC\_FLAG\_OUTPUT\_SPSPPS, 17
- NV\_ENC\_PIC\_STRUCT\_FIELD\_BOTTOM\_TOP, 18
- NV\_ENC\_PIC\_STRUCT\_FIELD\_TOP\_BOTTOM, 18
- NV\_ENC\_PIC\_STRUCT\_FRAME, 18
- NV\_ENC\_PIC\_TYPE\_B, 18
- NV\_ENC\_PIC\_TYPE\_BI, 18
- NV\_ENC\_PIC\_TYPE\_I, 18
- NV\_ENC\_PIC\_TYPE\_IDR, 18
- NV\_ENC\_PIC\_TYPE\_INTRA\_REFRESH, 18
- NV\_ENC\_PIC\_TYPE\_P, 18
- NV\_ENC\_PIC\_TYPE\_SKIPPED, 18
- NV\_ENC\_PIC\_TYPE\_UNKNOWN, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_CHECKERBOARD, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_COLINTERLEAVE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_FRAMESEQ, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_NONE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_ROWINTERLEAVE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_SIDE\_BY\_SIDE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_TOPBOTTOM, 18
- NV\_ENC\_SUCCESS, 18
- ENCODER\_STRUCTURE
  - NV\_ENC\_BUFFER\_FORMAT, 12
  - NV\_ENC\_CAPS, 12
  - NV\_ENC\_CAPS\_PARAM\_VER, 10
  - NV\_ENC\_CONFIG\_VER, 10
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER\_VER, 10
  - NV\_ENC\_CREATE\_INPUT\_BUFFER\_VER, 10
  - NV\_ENC\_CREATE\_MV\_BUFFER\_VER, 10
  - NV\_ENC\_DEVICE\_TYPE, 15
  - NV\_ENC\_EVENT\_PARAMS\_VER, 10
  - NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_MODE, 15

- NV\_ENC\_H264\_BDIRECT\_MODE, 15
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE, 15
- NV\_ENC\_H264\_FMO\_MODE, 16
- NV\_ENC\_HEVC\_CUSIZE, 16
- NV\_ENC\_INITIALIZE\_PARAMS\_VER, 11
- NV\_ENC\_INPUT\_RESOURCE\_TYPE, 16
- NV\_ENC\_LEVEL, 16
- NV\_ENC\_LOCK\_BITSTREAM\_VER, 11
- NV\_ENC\_LOCK\_INPUT\_BUFFER\_VER, 11
- NV\_ENC\_MAP\_INPUT\_RESOURCE\_VER, 11
- NV\_ENC\_MEMORY\_HEAP, 16
- NV\_ENC\_MEONLY\_PARAMS\_VER, 11
- NV\_ENC\_MV\_PRECISION, 16
- NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS\_VER, 11
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE, 17
- NV\_ENC\_PARAMS\_RC\_CBR2, 11
- NV\_ENC\_PARAMS\_RC\_MODE, 17
- NV\_ENC\_PIC\_FLAGS, 17
- NV\_ENC\_PIC\_PARAMS\_VER, 11
- NV\_ENC\_PIC\_STRUCT, 17
- NV\_ENC\_PIC\_TYPE, 18
- NV\_ENC\_PRESET\_CONFIG\_VER, 11
- NV\_ENC\_RC\_PARAMS\_VER, 11
- NV\_ENC\_RECONFIGURE\_PARAMS\_VER, 11
- NV\_ENC\_REGISTER\_RESOURCE\_VER, 12
- NV\_ENC\_SEQUENCE\_PARAM\_PAYLOAD\_VER, 12
- NV\_ENC\_STAT\_VER, 12
- NV\_ENC\_STEREO\_PACKING\_MODE, 18
- NVENCSTATUS, 18
  
- GUID, 67
  - Data1, 67
  - Data2, 67
  - Data3, 67
  - Data4, 67
  
- h264PicParams
  - NV\_ENC\_CODEC\_PIC\_PARAMS, 69
- height
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
- hevcPicParams
  - NV\_ENC\_CODEC\_PIC\_PARAMS, 69
  
- initialRCQP
  - NV\_ENC\_RC\_PARAMS, 82
- inputBuffer
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
  
- maxBitRate
  - NV\_ENC\_RC\_PARAMS, 82
- maxQP
  - NV\_ENC\_RC\_PARAMS, 82
  
- mb\_type
  - NV\_ENC\_H264\_MV\_DATA, 76
- memoryHeap
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 70
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
- minQP
  - NV\_ENC\_RC\_PARAMS, 82
- MV
  - NV\_ENC\_H264\_MV\_DATA, 76
- MVBuffer
  - NV\_ENC\_CREATE\_MV\_BUFFER, 74
- mvx
  - NV\_ENC\_MVECTOR, 77
- mvy
  - NV\_ENC\_MVECTOR, 77
  
- NV\_ENC\_BUFFER\_FORMAT\_ARGB\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_ARGB10\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_AYUV\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_IYUV\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_NV12\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_UNDEFINED\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_YUV444\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_BUFFER\_FORMAT\_YV12\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS\_ASYNC\_ENCODE\_SUPPORT\_ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_EXPOSED\_COUNT\_ENCODER\_STRUCTURE, 15
- NV\_ENC\_CAPS\_HEIGHT\_MAX\_ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_LEVEL\_MAX\_ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_LEVEL\_MIN\_ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_MB\_NUM\_MAX\_ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_MB\_PER\_SEC\_MAX\_ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_NUM\_MAX\_BFRAMES\_ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS\_NUM\_MAX\_TEMPORAL\_LAYERS\_ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_PREPROC\_SUPPORT\_ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SEPARATE\_COLOUR\_PLANE\_ENCODER\_STRUCTURE, 13

- NV\_ENC\_CAPS\_SUPPORT\_ADAPTIVE\_-  
TRANSFORM  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_BDIRECT\_MODE  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_CABAC  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_CONSTRAINED\_-  
ENCODING  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_CUSTOM\_VBV\_BUF\_-  
SIZE  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_BITRATE\_-  
CHANGE  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_FORCE\_-  
CONSTQP  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_RCMODE\_-  
CHANGE  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_DYN\_RES\_CHANGE  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_DYNAMIC\_SLICE\_-  
MODE  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_FIELD\_ENCODING  
ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS\_SUPPORT\_FMO  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_HIERARCHICAL\_-  
BFRAMES  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_HIERARCHICAL\_-  
PFRAMES  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_INTRA\_REFRESH  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_LOSSLESS\_ENCODE  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_CAPS\_SUPPORT\_MEONLY\_MODE  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_CAPS\_SUPPORT\_MONOCHROME  
ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS\_SUPPORT\_QPELMV  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_REF\_PIC\_-  
INVALIDATION  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_RESERVED  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_SAO  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_CAPS\_SUPPORT\_SUBFRAME\_-  
READBACK  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORT\_TEMPORAL\_SVC  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_CAPS\_SUPPORT\_YUV444\_ENCODE  
ENCODER\_STRUCTURE, 14
- NV\_ENC\_CAPS\_SUPPORTED\_RATECONTROL\_-  
MODES  
ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS\_WIDTH\_MAX  
ENCODER\_STRUCTURE, 13
- NV\_ENC\_DEVICE\_TYPE\_CUDA  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_DEVICE\_TYPE\_DIRECTX  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_ERR\_DEVICE\_NOT\_EXIST  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_ENCODER\_BUSY  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_ENCODER\_NOT\_INITIALIZED  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_EVENT\_NOT\_REGISTERD  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_GENERIC  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INCOMPATIBLE\_CLIENT\_KEY  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_CALL  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_DEVICE  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_ENCODERDEVICE  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_EVENT  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_PARAM  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_PTR  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_INVALID\_VERSION  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_LOCK\_BUSY  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_MAP\_FAILED  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_NEED\_MORE\_INPUT  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_NO\_ENCODE\_DEVICE  
ENCODER\_STRUCTURE, 18
- NV\_ENC\_ERR\_NOT\_ENOUGH\_BUFFER  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_OUT\_OF\_MEMORY  
ENCODER\_STRUCTURE, 19

- NV\_ENC\_ERR\_RESOURCE\_NOT\_MAPPED  
ENCODER\_STRUCTURE, 20
- NV\_ENC\_ERR\_RESOURCE\_NOT\_REGISTERED  
ENCODER\_STRUCTURE, 20
- NV\_ENC\_ERR\_RESOURCE\_REGISTER\_FAILED  
ENCODER\_STRUCTURE, 20
- NV\_ENC\_ERR\_UNIMPLEMENTED  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_UNSUPPORTED\_DEVICE  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_ERR\_UNSUPPORTED\_PARAM  
ENCODER\_STRUCTURE, 19
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_-  
AUTOSELECT  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_-  
DISABLE  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_-  
ENABLE  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_AUTOSELECT  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_DISABLE  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_SPATIAL  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE\_TEMPORAL  
ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_-  
AUTOSELECT  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_-  
CABAC  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE\_-  
CAVLC  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_H264\_FMO\_AUTOSELECT  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_H264\_FMO\_DISABLE  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_H264\_FMO\_ENABLE  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_CUDAARRAY  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_-  
CUDADEVICEPTR  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_INPUT\_RESOURCE\_TYPE\_DIRECTX  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_MEMORY\_HEAP\_AUTOSELECT  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_MEMORY\_HEAP\_SYSTEMEM\_CACHED  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_MEMORY\_HEAP\_SYSTEMEM\_UNCACHED  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_MEMORY\_HEAP\_VID  
ENCODER\_STRUCTURE, 16
- NV\_ENC\_MV\_PRECISION\_DEFAULT  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_MV\_PRECISION\_FULL\_PEL  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_MV\_PRECISION\_HALF\_PEL  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_MV\_PRECISION\_QUARTER\_PEL  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_FIELD  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_-  
FRAME  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE\_-  
MBAFF  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_2\_PASS\_FRAMESIZE\_CAP  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_2\_PASS\_QUALITY  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_2\_PASS\_VBR  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_CBR  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_CONSTQP  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_VBR  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_VBR\_MINQP  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_FLAG\_EOS  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_FLAG\_FORCEIDR  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_FLAG\_FORCEINTRA  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_FLAG\_OUTPUT\_SPSPPS  
ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_STRUCT\_FIELD\_BOTTOM\_TOP  
ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_STRUCT\_FIELD\_TOP\_BOTTOM  
ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_STRUCT\_FRAME  
ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_B  
ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_BI  
ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_I



- ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_IDR
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_INTRA\_REFRESH
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_P
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_SKIPPED
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_PIC\_TYPE\_UNKNOWN
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_-CHECKERBOARD
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_-COLINTERLEAVE
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_FRAMESEQ
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_NONE
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_-ROWINTERLEAVE
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_SIDE BYSIDE
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_STEREO\_PACKING\_MODE\_-TOPBOTTOM
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_SUCCESS
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_BUFFER\_FORMAT
  - ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS
  - ENCODER\_STRUCTURE, 12
- NV\_ENC\_CAPS\_PARAM, 68
  - capsToQuery, 68
  - reserved, 68
  - version, 68
- NV\_ENC\_CAPS\_PARAM\_VER
  - ENCODER\_STRUCTURE, 10
- NV\_ENC\_CODEC\_PIC\_PARAMS, 69
  - h264PicParams, 69
  - hevcPicParams, 69
  - reserved, 69
- NV\_ENC\_CONFIG\_VER
  - ENCODER\_STRUCTURE, 10
- NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 70
  - bitstreamBuffer, 70
  - bitstreamBufferPtr, 70
  - memoryHeap, 70
  - reserved, 70
  - reserved1, 70
  - reserved2, 70
  - size, 70
  - version, 71
- NV\_ENC\_CREATE\_BITSTREAM\_BUFFER\_VER
  - ENCODER\_STRUCTURE, 10
- NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
  - bufferFmt, 72
  - height, 72
  - inputBuffer, 72
  - memoryHeap, 72
  - pSysMemBuffer, 72
  - reserved, 72
  - reserved1, 72
  - reserved2, 73
  - version, 73
  - width, 73
- NV\_ENC\_CREATE\_INPUT\_BUFFER\_VER
  - ENCODER\_STRUCTURE, 10
- NV\_ENC\_CREATE\_MV\_BUFFER, 74
  - MVBuffer, 74
  - reserved1, 74
  - reserved2, 74
  - version, 74
- NV\_ENC\_CREATE\_MV\_BUFFER\_VER
  - ENCODER\_STRUCTURE, 10
- NV\_ENC\_DEVICE\_TYPE
  - ENCODER\_STRUCTURE, 15
- NV\_ENC\_EVENT\_PARAMS, 75
  - completionEvent, 75
  - reserved, 75
  - reserved1, 75
  - reserved2, 75
  - version, 75
- NV\_ENC\_EVENT\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 10
- NV\_ENC\_H264\_ADAPTIVE\_TRANSFORM\_MODE
  - ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_BDIRECT\_MODE
  - ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_ENTROPY\_CODING\_MODE
  - ENCODER\_STRUCTURE, 15
- NV\_ENC\_H264\_FMO\_MODE
  - ENCODER\_STRUCTURE, 16
- NV\_ENC\_H264\_MV\_DATA, 76
  - mb\_type, 76
  - MV, 76
  - partitionType, 76
  - reserved, 76
- NV\_ENC\_HEVC\_CUSIZE
  - ENCODER\_STRUCTURE, 16
- NV\_ENC\_INITIALIZE\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_INPUT\_RESOURCE\_TYPE
  - ENCODER\_STRUCTURE, 16
- NV\_ENC\_LEVEL

- ENCODER\_STRUCTURE, 16
- NV\_ENC\_LOCK\_BITSTREAM\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_LOCK\_INPUT\_BUFFER\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_MAP\_INPUT\_RESOURCE\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_MEMORY\_HEAP
  - ENCODER\_STRUCTURE, 16
- NV\_ENC\_MEONLY\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_MV\_PRECISION
  - ENCODER\_STRUCTURE, 16
- NV\_ENC\_MVECTOR, 77
  - mvx, 77
  - my, 77
- NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS, 78
  - apiVersion, 78
  - device, 78
  - deviceType, 78
  - reserved, 78
  - reserved1, 78
  - reserved2, 78
  - version, 78
- NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_PARAMS\_FRAME\_FIELD\_MODE
  - ENCODER\_STRUCTURE, 17
- NV\_ENC\_PARAMS\_RC\_CBR2
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_PARAMS\_RC\_MODE
  - ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_FLAGS
  - ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_PIC\_STRUCT
  - ENCODER\_STRUCTURE, 17
- NV\_ENC\_PIC\_TYPE
  - ENCODER\_STRUCTURE, 18
- NV\_ENC\_PRESET\_CONFIG\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_QP, 80
- NV\_ENC\_RC\_PARAMS, 81
  - averageBitRate, 81
  - constQP, 81
  - enableAQ, 81
  - enableExtQPDeltaMap, 81
  - enableInitialRCQP, 81
  - enableMaxQP, 82
  - enableMinQP, 82
  - initialRCQP, 82
  - maxBitRate, 82
  - maxQP, 82
  - minQP, 82
  - rateControlMode, 82
  - reservedBitFields, 82
  - temporalLayerIdxMask, 82
  - temporalLayerQP, 82
  - vbvBufferSize, 82
  - vbvInitialDelay, 83
- NV\_ENC\_RC\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_RECONFIGURE\_PARAMS\_VER
  - ENCODER\_STRUCTURE, 11
- NV\_ENC\_REGISTER\_RESOURCE\_VER
  - ENCODER\_STRUCTURE, 12
- NV\_ENC\_SEQUENCE\_PARAM\_PAYLOAD\_VER
  - ENCODER\_STRUCTURE, 12
- NV\_ENC\_STAT\_VER
  - ENCODER\_STRUCTURE, 12
- NV\_ENC\_STEREO\_PACKING\_MODE
  - ENCODER\_STRUCTURE, 18
- NV\_ENCODE\_API\_FUNCTION\_LIST, 84
  - nvEncCreateBitstreamBuffer, 85
  - nvEncCreateInputBuffer, 85
  - nvEncCreateMVBuffer, 85
  - nvEncDestroyBitstreamBuffer, 85
  - nvEncDestroyEncoder, 85
  - nvEncDestroyInputBuffer, 85
  - nvEncDestroyMVBuffer, 85
  - nvEncEncodePicture, 85
  - nvEncGetEncodeCaps, 85
  - nvEncGetEncodeGUIDCount, 85
  - nvEncGetEncodeGUIDs, 85
  - nvEncGetEncodePresetConfig, 86
  - nvEncGetEncodePresetCount, 86
  - nvEncGetEncodePresetGUIDs, 86
  - nvEncGetEncodeProfileGUIDCount, 86
  - nvEncGetEncodeProfileGUIDs, 86
  - nvEncGetEncodeStats, 86
  - nvEncGetInputFormatCount, 86
  - nvEncGetInputFormats, 86
  - nvEncGetSequenceParams, 86
  - nvEncInitializeEncoder, 86
  - nvEncInvalidateRefFrames, 87
  - nvEncLockBitstream, 87
  - nvEncLockInputBuffer, 87
  - nvEncMapInputResource, 87
  - nvEncOpenEncodeSession, 87
  - nvEncOpenEncodeSessionEx, 87
  - nvEncReconfigureEncoder, 87
  - nvEncRegisterAsyncEvent, 87
  - nvEncRegisterResource, 87
  - nvEncRunMotionEstimationOnly, 87
  - nvEncUnlockBitstream, 88

- nvEncUnlockInputBuffer, 88
- nvEncUnmapInputResource, 88
- nvEncUnregisterAsyncEvent, 88
- nvEncUnregisterResource, 88
- reserved, 88
- reserved2, 88
- version, 88
- NvEncCreateBitstreamBuffer
  - ENCODE\_FUNC, 24
- nvEncCreateBitstreamBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncCreateInputBuffer
  - ENCODE\_FUNC, 24
- nvEncCreateInputBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncCreateMVBBuffer
  - ENCODE\_FUNC, 24
- nvEncCreateMVBBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncDestroyBitstreamBuffer
  - ENCODE\_FUNC, 25
- nvEncDestroyBitstreamBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncDestroyEncoder
  - ENCODE\_FUNC, 25
- nvEncDestroyEncoder
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncDestroyInputBuffer
  - ENCODE\_FUNC, 26
- nvEncDestroyInputBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncDestroyMVBBuffer
  - ENCODE\_FUNC, 26
- nvEncDestroyMVBBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncEncodePicture
  - ENCODE\_FUNC, 27
- nvEncEncodePicture
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncGetEncodeCaps
  - ENCODE\_FUNC, 29
- nvEncGetEncodeCaps
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncGetEncodeGUIDCount
  - ENCODE\_FUNC, 30
- nvEncGetEncodeGUIDCount
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncGetEncodeGUIDs
  - ENCODE\_FUNC, 30
- nvEncGetEncodeGUIDs
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 85
- NvEncGetEncodePresetConfig
  - ENCODE\_FUNC, 31
- nvEncGetEncodePresetConfig
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetEncodePresetCount
  - ENCODE\_FUNC, 31
- nvEncGetEncodePresetCount
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetEncodePresetGUIDs
  - ENCODE\_FUNC, 32
- nvEncGetEncodePresetGUIDs
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetEncodeProfileGUIDCount
  - ENCODE\_FUNC, 32
- nvEncGetEncodeProfileGUIDCount
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetEncodeProfileGUIDs
  - ENCODE\_FUNC, 33
- nvEncGetEncodeProfileGUIDs
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetEncodeStats
  - ENCODE\_FUNC, 33
- nvEncGetEncodeStats
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetInputFormatCount
  - ENCODE\_FUNC, 34
- nvEncGetInputFormatCount
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetInputFormats
  - ENCODE\_FUNC, 34
- nvEncGetInputFormats
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncGetSequenceParams
  - ENCODE\_FUNC, 35
- nvEncGetSequenceParams
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncInitializeEncoder
  - ENCODE\_FUNC, 35
- nvEncInitializeEncoder
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 86
- NvEncInvalidateRefFrames
  - ENCODE\_FUNC, 37
- nvEncInvalidateRefFrames
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncLockBitstream
  - ENCODE\_FUNC, 37
- nvEncLockBitstream
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncLockInputBuffer
  - ENCODE\_FUNC, 38
- nvEncLockInputBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncMapInputResource
  - ENCODE\_FUNC, 38
- nvEncMapInputResource
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncodeAPI Data structures, 7

- NvEncodeAPI Functions, 21
- NvEncodeAPICreateInstance
  - ENCODE\_FUNC, 39
- NvEncOpenEncodeSession
  - ENCODE\_FUNC, 39
- nvEncOpenEncodeSession
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncOpenEncodeSessionEx
  - ENCODE\_FUNC, 39
- nvEncOpenEncodeSessionEx
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncReconfigureEncoder
  - ENCODE\_FUNC, 40
- nvEncReconfigureEncoder
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncRegisterAsyncEvent
  - ENCODE\_FUNC, 40
- nvEncRegisterAsyncEvent
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncRegisterResource
  - ENCODE\_FUNC, 41
- nvEncRegisterResource
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NvEncRunMotionEstimationOnly
  - ENCODE\_FUNC, 41
- nvEncRunMotionEstimationOnly
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 87
- NVENCSTATUS
  - ENCODER\_STRUCTURE, 18
- NvEncUnlockBitstream
  - ENCODE\_FUNC, 42
- nvEncUnlockBitstream
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- NvEncUnlockInputBuffer
  - ENCODE\_FUNC, 42
- nvEncUnlockInputBuffer
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- NvEncUnmapInputResource
  - ENCODE\_FUNC, 43
- nvEncUnmapInputResource
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- NvEncUnregisterAsyncEvent
  - ENCODE\_FUNC, 43
- nvEncUnregisterAsyncEvent
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- NvEncUnregisterResource
  - ENCODE\_FUNC, 44
- nvEncUnregisterResource
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- partitionType
  - NV\_ENC\_H264\_MV\_DATA, 76
- pSysMemBuffer
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
- rateControlMode
  - NV\_ENC\_RC\_PARAMS, 82
- reserved
  - NV\_ENC\_CAPS\_PARAM, 68
  - NV\_ENC\_CODEC\_PIC\_PARAMS, 69
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 70
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
  - NV\_ENC\_EVENT\_PARAMS, 75
  - NV\_ENC\_H264\_MV\_DATA, 76
  - NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS, 78
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- reserved1
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 70
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 72
  - NV\_ENC\_CREATE\_MV\_BUFFER, 74
  - NV\_ENC\_EVENT\_PARAMS, 75
  - NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS, 78
- reserved2
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 70
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 73
  - NV\_ENC\_CREATE\_MV\_BUFFER, 74
  - NV\_ENC\_EVENT\_PARAMS, 75
  - NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS, 78
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- reservedBitFields
  - NV\_ENC\_RC\_PARAMS, 82
- size
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 70
- temporalLayerIdxMask
  - NV\_ENC\_RC\_PARAMS, 82
- temporalLayerQP
  - NV\_ENC\_RC\_PARAMS, 82
- vbvBufferSize
  - NV\_ENC\_RC\_PARAMS, 82
- vbvInitialDelay
  - NV\_ENC\_RC\_PARAMS, 83
- version
  - NV\_ENC\_CAPS\_PARAM, 68
  - NV\_ENC\_CREATE\_BITSTREAM\_BUFFER, 71
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 73
  - NV\_ENC\_CREATE\_MV\_BUFFER, 74
  - NV\_ENC\_EVENT\_PARAMS, 75
  - NV\_ENC\_OPEN\_ENCODE\_SESSION\_EX\_PARAMS, 78
  - NV\_ENCODE\_API\_FUNCTION\_LIST, 88
- width
  - NV\_ENC\_CREATE\_INPUT\_BUFFER, 73

### **Notice**

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

### **Trademarks**

NVIDIA, the NVIDIA logo, GeForce, Tesla, and Quadro are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

### **Copyright**

© 2007-2012 NVIDIA Corporation. All rights reserved.