



NVENC - NVIDIA Video Encoder API Reference Manual

January 27, 2015

Version 5.0



Contents

| | | |
|----------|--|----------|
| 1 | Legal Notice | 1 |
| 2 | Module Index | 3 |
| 2.1 | Modules | 3 |
| 3 | Data Structure Index | 5 |
| 3.1 | Data Structures | 5 |
| 4 | Module Documentation | 7 |
| 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 | 11 |
| 4.1.1.5 | NV_ENC_EVENT_PARAMS_VER | 11 |
| 4.1.1.6 | NV_ENC_INITIALIZE_PARAMS_VER | 11 |
| 4.1.1.7 | NV_ENC_LOCK_BITSTREAM_VER | 11 |
| 4.1.1.8 | NV_ENC_LOCK_INPUT_BUFFER_VER | 11 |
| 4.1.1.9 | NV_ENC_MAP_INPUT_RESOURCE_VER | 11 |
| 4.1.1.10 | NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER | 11 |
| 4.1.1.11 | NV_ENC_PARAMS_RC_CBR2 | 11 |
| 4.1.1.12 | NV_ENC_PIC_PARAMS_VER | 11 |
| 4.1.1.13 | NV_ENC_PRESET_CONFIG_VER | 12 |
| 4.1.1.14 | NV_ENC_RC_PARAMS_VER | 12 |
| 4.1.1.15 | NV_ENC_RECONFIGURE_PARAMS_VER | 12 |
| 4.1.1.16 | NV_ENC_REGISTER_RESOURCE_VER | 12 |
| 4.1.1.17 | NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER | 12 |
| 4.1.1.18 | 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 | 13 |
| 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 | 16 |
| 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 | 17 |
| 4.1.2.11 | NV_ENC_MEMORY_HEAP | 17 |
| 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 | 18 |
| 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 | 19 |
| 4.1.2.19 | NVENCSTATUS | 19 |
| 4.2 | NvEncodeAPI Functions | 21 |
| 4.2.1 | Function Documentation | 23 |
| 4.2.1.1 | NvEncCreateBitstreamBuffer | 23 |
| 4.2.1.2 | NvEncCreateInputBuffer | 24 |
| 4.2.1.3 | NvEncDestroyBitstreamBuffer | 24 |
| 4.2.1.4 | NvEncDestroyEncoder | 25 |
| 4.2.1.5 | NvEncDestroyInputBuffer | 25 |
| 4.2.1.6 | NvEncEncodePicture | 26 |
| 4.2.1.7 | NvEncGetEncodeCaps | 28 |
| 4.2.1.8 | NvEncGetEncodeGUIDCount | 29 |
| 4.2.1.9 | NvEncGetEncodeGUIDs | 29 |
| 4.2.1.10 | NvEncGetEncodePresetConfig | 30 |
| 4.2.1.11 | NvEncGetEncodePresetCount | 30 |
| 4.2.1.12 | NvEncGetEncodePresetGUIDs | 31 |
| 4.2.1.13 | NvEncGetEncodeProfileGUIDCount | 31 |
| 4.2.1.14 | NvEncGetEncodeProfileGUIDs | 32 |
| 4.2.1.15 | NvEncGetEncodeStats | 32 |

| | | |
|----------|---|-----------|
| 4.2.1.16 | NvEncGetInputFormatCount | 33 |
| 4.2.1.17 | NvEncGetInputFormats | 33 |
| 4.2.1.18 | NvEncGetSequenceParams | 34 |
| 4.2.1.19 | NvEncInitializeEncoder | 34 |
| 4.2.1.20 | NvEncInvalidateRefFrames | 36 |
| 4.2.1.21 | NvEncLockBitstream | 36 |
| 4.2.1.22 | NvEncLockInputBuffer | 37 |
| 4.2.1.23 | NvEncMapInputResource | 37 |
| 4.2.1.24 | NvEncodeAPICreateInstance | 38 |
| 4.2.1.25 | NvEncOpenEncodeSession | 38 |
| 4.2.1.26 | NvEncOpenEncodeSessionEx | 39 |
| 4.2.1.27 | NvEncReconfigureEncoder | 39 |
| 4.2.1.28 | NvEncRegisterAsyncEvent | 39 |
| 4.2.1.29 | NvEncRegisterResource | 40 |
| 4.2.1.30 | NvEncUnlockBitstream | 40 |
| 4.2.1.31 | NvEncUnlockInputBuffer | 41 |
| 4.2.1.32 | NvEncUnmapInputResource | 41 |
| 4.2.1.33 | NvEncUnregisterAsyncEvent | 42 |
| 4.2.1.34 | NvEncUnregisterResource | 42 |
| 5 | Data Structure Documentation | 45 |
| 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_H264_SEI_PAYLOAD Struct Reference | 50 |
| 5.6.1 | Detailed Description | 50 |
| 5.7 | _NV_ENC_INITIALIZE_PARAMS Struct Reference | 51 |
| 5.7.1 | Detailed Description | 51 |
| 5.8 | _NV_ENC_LOCK_BITSTREAM Struct Reference | 52 |
| 5.8.1 | Detailed Description | 52 |

| | | |
|----------|---|----|
| 5.9 | _NV_ENC_LOCK_INPUT_BUFFER Struct Reference | 53 |
| 5.9.1 | Detailed Description | 53 |
| 5.10 | _NV_ENC_MAP_INPUT_RESOURCE Struct Reference | 54 |
| 5.10.1 | Detailed Description | 54 |
| 5.11 | _NV_ENC_PIC_PARAMS Struct Reference | 55 |
| 5.11.1 | Detailed Description | 55 |
| 5.12 | _NV_ENC_PIC_PARAMS_H264 Struct Reference | 56 |
| 5.12.1 | Detailed Description | 56 |
| 5.13 | _NV_ENC_PIC_PARAMS_HEVC Struct Reference | 57 |
| 5.13.1 | Detailed Description | 57 |
| 5.14 | _NV_ENC_PRESET_CONFIG Struct Reference | 58 |
| 5.14.1 | Detailed Description | 58 |
| 5.15 | _NV_ENC_RECONFIGURE_PARAMS Struct Reference | 59 |
| 5.15.1 | Detailed Description | 59 |
| 5.16 | _NV_ENC_REGISTER_RESOURCE Struct Reference | 60 |
| 5.16.1 | Detailed Description | 60 |
| 5.17 | _NV_ENC_SEQUENCE_PARAM_PAYLOAD Struct Reference | 61 |
| 5.17.1 | Detailed Description | 61 |
| 5.18 | _NV_ENC_STAT Struct Reference | 62 |
| 5.18.1 | Detailed Description | 62 |
| 5.19 | _NVENC_EXTERNAL_ME_HINT Struct Reference | 63 |
| 5.19.1 | Detailed Description | 63 |
| 5.20 | _NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE Struct Reference | 64 |
| 5.20.1 | Detailed Description | 64 |
| 5.21 | _NVENC_RECT Struct Reference | 65 |
| 5.21.1 | Detailed Description | 65 |
| 5.22 | GUID Struct Reference | 66 |
| 5.22.1 | Detailed Description | 66 |
| 5.22.2 | Field Documentation | 66 |
| 5.22.2.1 | Data1 | 66 |
| 5.22.2.2 | Data2 | 66 |
| 5.22.2.3 | Data3 | 66 |
| 5.22.2.4 | Data4 | 66 |
| 5.23 | NV_ENC_CAPS_PARAM Struct Reference | 67 |
| 5.23.1 | Detailed Description | 67 |
| 5.23.2 | Field Documentation | 67 |
| 5.23.2.1 | capsToQuery | 67 |

| | | |
|-----------|---|----|
| 5.23.2.2 | reserved | 67 |
| 5.23.2.3 | version | 67 |
| 5.24 | NV_ENC_CODEC_PIC_PARAMS Union Reference | 68 |
| 5.24.1 | Detailed Description | 68 |
| 5.24.2 | Field Documentation | 68 |
| 5.24.2.1 | h264PicParams | 68 |
| 5.24.2.2 | hevcPicParams | 68 |
| 5.24.2.3 | reserved | 68 |
| 5.25 | NV_ENC_CREATE_BITSTREAM_BUFFER Struct Reference | 69 |
| 5.25.1 | Detailed Description | 69 |
| 5.25.2 | Field Documentation | 69 |
| 5.25.2.1 | bitstreamBuffer | 69 |
| 5.25.2.2 | bitstreamBufferPtr | 69 |
| 5.25.2.3 | memoryHeap | 69 |
| 5.25.2.4 | reserved | 69 |
| 5.25.2.5 | reserved1 | 69 |
| 5.25.2.6 | reserved2 | 69 |
| 5.25.2.7 | size | 70 |
| 5.25.2.8 | version | 70 |
| 5.26 | NV_ENC_CREATE_INPUT_BUFFER Struct Reference | 71 |
| 5.26.1 | Detailed Description | 71 |
| 5.26.2 | Field Documentation | 71 |
| 5.26.2.1 | bufferFmt | 71 |
| 5.26.2.2 | height | 71 |
| 5.26.2.3 | inputBuffer | 71 |
| 5.26.2.4 | memoryHeap | 71 |
| 5.26.2.5 | pSysMemBuffer | 71 |
| 5.26.2.6 | reserved | 71 |
| 5.26.2.7 | reserved1 | 72 |
| 5.26.2.8 | reserved2 | 72 |
| 5.26.2.9 | version | 72 |
| 5.26.2.10 | width | 72 |
| 5.27 | NV_ENC_EVENT_PARAMS Struct Reference | 73 |
| 5.27.1 | Detailed Description | 73 |
| 5.27.2 | Field Documentation | 73 |
| 5.27.2.1 | completionEvent | 73 |
| 5.27.2.2 | reserved | 73 |

| | | |
|-----------|---|----|
| 5.27.2.3 | reserved1 | 73 |
| 5.27.2.4 | reserved2 | 73 |
| 5.27.2.5 | version | 73 |
| 5.28 | NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS Struct Reference | 74 |
| 5.28.1 | Detailed Description | 74 |
| 5.28.2 | Field Documentation | 74 |
| 5.28.2.1 | apiVersion | 74 |
| 5.28.2.2 | device | 74 |
| 5.28.2.3 | deviceType | 74 |
| 5.28.2.4 | reserved | 74 |
| 5.28.2.5 | reserved1 | 74 |
| 5.28.2.6 | reserved2 | 74 |
| 5.28.2.7 | version | 75 |
| 5.29 | NV_ENC_QP Struct Reference | 76 |
| 5.29.1 | Detailed Description | 76 |
| 5.30 | NV_ENC_RC_PARAMS Struct Reference | 77 |
| 5.30.1 | Detailed Description | 77 |
| 5.30.2 | Field Documentation | 77 |
| 5.30.2.1 | averageBitRate | 77 |
| 5.30.2.2 | constQP | 77 |
| 5.30.2.3 | enableAQ | 77 |
| 5.30.2.4 | enableExtQPDeltaMap | 77 |
| 5.30.2.5 | enableInitialRCQP | 78 |
| 5.30.2.6 | enableMaxQP | 78 |
| 5.30.2.7 | enableMinQP | 78 |
| 5.30.2.8 | initialRCQP | 78 |
| 5.30.2.9 | maxBitRate | 78 |
| 5.30.2.10 | maxQP | 78 |
| 5.30.2.11 | minQP | 78 |
| 5.30.2.12 | rateControlMode | 78 |
| 5.30.2.13 | reservedBitFields | 78 |
| 5.30.2.14 | temporalLayerIdxMask | 78 |
| 5.30.2.15 | temporalLayerQP | 78 |
| 5.30.2.16 | vbvBufferSize | 79 |
| 5.30.2.17 | vbvInitialDelay | 79 |
| 5.31 | NV_ENCODE_API_FUNCTION_LIST Struct Reference | 80 |
| 5.31.1 | Detailed Description | 80 |

| | | |
|-----------|--------------------------------|----|
| 5.31.2 | Field Documentation | 81 |
| 5.31.2.1 | nvEncCreateBitstreamBuffer | 81 |
| 5.31.2.2 | nvEncCreateInputBuffer | 81 |
| 5.31.2.3 | nvEncDestroyBitstreamBuffer | 81 |
| 5.31.2.4 | nvEncDestroyEncoder | 81 |
| 5.31.2.5 | nvEncDestroyInputBuffer | 81 |
| 5.31.2.6 | nvEncEncodePicture | 81 |
| 5.31.2.7 | nvEncGetEncodeCaps | 81 |
| 5.31.2.8 | nvEncGetEncodeGUIDCount | 81 |
| 5.31.2.9 | nvEncGetEncodeGUIDs | 81 |
| 5.31.2.10 | nvEncGetEncodePresetConfig | 81 |
| 5.31.2.11 | nvEncGetEncodePresetCount | 82 |
| 5.31.2.12 | nvEncGetEncodePresetGUIDs | 82 |
| 5.31.2.13 | nvEncGetEncodeProfileGUIDCount | 82 |
| 5.31.2.14 | nvEncGetEncodeProfileGUIDs | 82 |
| 5.31.2.15 | nvEncGetEncodeStats | 82 |
| 5.31.2.16 | nvEncGetInputFormatCount | 82 |
| 5.31.2.17 | nvEncGetInputFormats | 82 |
| 5.31.2.18 | nvEncGetSequenceParams | 82 |
| 5.31.2.19 | nvEncInitializeEncoder | 82 |
| 5.31.2.20 | nvEncInvalidateRefFrames | 82 |
| 5.31.2.21 | nvEncLockBitstream | 83 |
| 5.31.2.22 | nvEncLockInputBuffer | 83 |
| 5.31.2.23 | nvEncMapInputResource | 83 |
| 5.31.2.24 | nvEncOpenEncodeSession | 83 |
| 5.31.2.25 | nvEncOpenEncodeSessionEx | 83 |
| 5.31.2.26 | nvEncReconfigureEncoder | 83 |
| 5.31.2.27 | nvEncRegisterAsyncEvent | 83 |
| 5.31.2.28 | nvEncRegisterResource | 83 |
| 5.31.2.29 | nvEncUnlockBitstream | 83 |
| 5.31.2.30 | nvEncUnlockInputBuffer | 83 |
| 5.31.2.31 | nvEncUnmapInputResource | 84 |
| 5.31.2.32 | nvEncUnregisterAsyncEvent | 84 |
| 5.31.2.33 | nvEncUnregisterResource | 84 |
| 5.31.2.34 | reserved | 84 |
| 5.31.2.35 | reserved2 | 84 |
| 5.31.2.36 | version | 84 |

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:

| | |
|--|----|
| _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_H264_SEI_PAYLOAD | 50 |
| _NV_ENC_INITIALIZE_PARAMS | 51 |
| _NV_ENC_LOCK_BITSTREAM | 52 |
| _NV_ENC_LOCK_INPUT_BUFFER | 53 |
| _NV_ENC_MAP_INPUT_RESOURCE | 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_SEQUENCE_PARAM_PAYLOAD | 61 |
| _NV_ENC_STAT | 62 |
| _NVENC_EXTERNAL_ME_HINT | 63 |
| _NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE | 64 |
| _NVENC_RECT | 65 |
| GUID | 66 |
| NV_ENC_CAPS_PARAM | 67 |
| NV_ENC_CODEC_PIC_PARAMS | 68 |
| NV_ENC_CREATE_BITSTREAM_BUFFER | 69 |
| NV_ENC_CREATE_INPUT_BUFFER | 71 |
| NV_ENC_EVENT_PARAMS | 73 |
| NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS | 74 |
| NV_ENC_QP | 76 |
| NV_ENC_RC_PARAMS | 77 |
| NV_ENCODE_API_FUNCTION_LIST | 80 |

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_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_H264_SEI_PAYLOAD](#)
- struct [_NV_ENC_PIC_PARAMS_H264](#)
- struct [_NV_ENC_PIC_PARAMS_HEVC](#)
- struct [_NV_ENC_PIC_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(NV_ENC_CAPS_PARAM, 1)`
- `#define NV_ENC_CREATE_INPUT_BUFFER_VER NVENCAPI_STRUCT_VERSION(NV_ENC_CREATE_INPUT_BUFFER, 1)`
- `#define NV_ENC_CREATE_BITSTREAM_BUFFER_VER NVENCAPI_STRUCT_VERSION(NV_ENC_CREATE_BITSTREAM_BUFFER, 1)`
- `#define NV_ENC_RC_PARAMS_VER NVENCAPI_STRUCT_VERSION(NV_ENC_RC_PARAMS, 1)`
- `#define NV_ENC_CONFIG_VER (NVENCAPI_STRUCT_VERSION(NV_ENC_CONFIG, 5) | (1 << 31))`
- `#define NV_ENC_INITIALIZE_PARAMS_VER (NVENCAPI_STRUCT_VERSION(NV_ENC_INITIALIZE_PARAMS, 5) | (1 << 31))`
- `#define NV_ENC_RECONFIGURE_PARAMS_VER (NVENCAPI_STRUCT_VERSION(NV_ENC_RECONFIGURE_PARAMS, 1) | (1 << 31))`
- `#define NV_ENC_PRESET_CONFIG_VER (NVENCAPI_STRUCT_VERSION(NV_ENC_PRESET_CONFIG, 4) | (1 << 31))`
- `#define NV_ENC_PIC_PARAMS_VER (NVENCAPI_STRUCT_VERSION(NV_ENC_PIC_PARAMS, 4) | (1 << 31))`
- `#define NV_ENC_LOCK_BITSTREAM_VER NVENCAPI_STRUCT_VERSION(NV_ENC_LOCK_BITSTREAM, 1)`
- `#define NV_ENC_LOCK_INPUT_BUFFER_VER NVENCAPI_STRUCT_VERSION(NV_ENC_LOCK_INPUT_BUFFER, 1)`
- `#define NV_ENC_MAP_INPUT_RESOURCE_VER NVENCAPI_STRUCT_VERSION(NV_ENC_MAP_INPUT_RESOURCE, 4)`
- `#define NV_ENC_REGISTER_RESOURCE_VER NVENCAPI_STRUCT_VERSION(NV_ENC_REGISTER_RESOURCE, 3)`
- `#define NV_ENC_STAT_VER NVENCAPI_STRUCT_VERSION(NV_ENC_STAT, 1)`
- `#define NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER NVENCAPI_STRUCT_VERSION(NV_ENC_SEQUENCE_PARAM_PAYLOAD, 1)`
- `#define NV_ENC_EVENT_PARAMS_VER NVENCAPI_STRUCT_VERSION(NV_ENC_EVENT_PARAMS, 1)`
- `#define NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER NVENCAPI_STRUCT_VERSION(NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 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_PL = 0x1, NV_ENC_BUFFER_FORMAT_NV12_TILED16x16 = 0x2, NV_ENC_BUFFER_FORMAT_NV12_TILED64x16 = 0x3,
 NV_ENC_BUFFER_FORMAT_YV12_PL = 0x10, NV_ENC_BUFFER_FORMAT_YV12_TILED16x16 = 0x20, NV_ENC_BUFFER_FORMAT_YV12_TILED64x16 = 0x30, NV_ENC_BUFFER_FORMAT_IYUV_PL = 0x100,
 NV_ENC_BUFFER_FORMAT_IYUV_TILED16x16 = 0x200, NV_ENC_BUFFER_FORMAT_IYUV_TILED64x16 = 0x300, NV_ENC_BUFFER_FORMAT_YUV444_PL = 0x1000, NV_ENC_BUFFER_FORMAT_YUV444_TILED16x16 = 0x2000,
 NV_ENC_BUFFER_FORMAT_YUV444_TILED64x16 = 0x3000 }
- 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_SPSPS = 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_SYSMEM_CACHED = 2, NV_ENC_MEMORY_HEAP_SYSMEM_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 BYSIDE = 0x4, NV_ENC_STEREO_PACKING_MODE_-
TOPBOTTOM = 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_EXPOSED_COUNT }

```
  - enum `NV_ENC_HEVC_CUSIZE`

#### 4.1.1 Define Documentation

##### 4.1.1.1 `#define NV_ENC_CAPS_PARAM_VER NVENCAPI_STRUCT_VERSION(NV_ENC_CAPS_-PARAM, 1)`

`NV_ENC_CAPS_PARAM` struct version.

##### 4.1.1.2 `#define NV_ENC_CONFIG_VER (NVENCAPI_STRUCT_VERSION(NV_ENC_CONFIG, 5) | (1 << 31))`

macro for constructing the version field of `_NV_ENC_CONFIG`

##### 4.1.1.3 `#define NV_ENC_CREATE_BITSTREAM_BUFFER_VER NVENCAPI_STRUCT_VERSION(NV_-ENC_CREATE_BITSTREAM_BUFFER, 1)`

`NV_ENC_CREATE_BITSTREAM_BUFFER` struct version.

**4.1.1.4** `#define NV_ENC_CREATE_INPUT_BUFFER_VER NVENC_API_STRUCT_VERSION(NV_ENC_CREATE_INPUT_BUFFER, 1)`

[NV\\_ENC\\_CREATE\\_INPUT\\_BUFFER](#) struct version.

**4.1.1.5** `#define NV_ENC_EVENT_PARAMS_VER NVENC_API_STRUCT_VERSION(NV_ENC_EVENT_PARAMS, 1)`

Macro for constructing the version field of [\\_NV\\_ENC\\_EVENT\\_PARAMS](#)

**4.1.1.6** `#define NV_ENC_INITIALIZE_PARAMS_VER (NVENC_API_STRUCT_VERSION(NV_ENC_INITIALIZE_PARAMS, 5) | (1 < 31))`

macro for constructing the version field of [\\_NV\\_ENC\\_INITIALIZE\\_PARAMS](#)

**4.1.1.7** `#define NV_ENC_LOCK_BITSTREAM_VER NVENC_API_STRUCT_VERSION(NV_ENC_LOCK_BITSTREAM, 1)`

Macro for constructing the version field of [\\_NV\\_ENC\\_LOCK\\_BITSTREAM](#)

**4.1.1.8** `#define NV_ENC_LOCK_INPUT_BUFFER_VER NVENC_API_STRUCT_VERSION(NV_ENC_LOCK_INPUT_BUFFER, 1)`

Macro for constructing the version field of [\\_NV\\_ENC\\_LOCK\\_INPUT\\_BUFFER](#)

**4.1.1.9** `#define NV_ENC_MAP_INPUT_RESOURCE_VER NVENC_API_STRUCT_VERSION(NV_ENC_MAP_INPUT_RESOURCE, 4)`

Macro for constructing the version field of [\\_NV\\_ENC\\_MAP\\_INPUT\\_RESOURCE](#)

**4.1.1.10** `#define NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER NVENC_API_STRUCT_VERSION(NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 1)`

Macro for constructing the version field of [\\_NV\\_ENC\\_OPEN\\_ENCODE\\_SESSIONEX\\_PARAMS](#)

**4.1.1.11** `#define NV_ENC_PARAMS_RC_CBR2 NV_ENC_PARAMS_RC_CBR`

Deprecated

**4.1.1.12** `#define NV_ENC_PIC_PARAMS_VER (NVENC_API_STRUCT_VERSION(NV_ENC_PIC_PARAMS, 4) | (1 < 31))`

Macro for constructing the version field of [\\_NV\\_ENC\\_PIC\\_PARAMS](#)

**4.1.1.13** `#define NV_ENC_PRESET_CONFIG_VER (NVENC_API_STRUCT_VERSION(NV_ENC_PRESET_CONFIG, 4) | (1 < 31))`

macro for constructing the version field of [\\_NV\\_ENC\\_PRESET\\_CONFIG](#)

**4.1.1.14** `#define NV_ENC_RC_PARAMS_VER NVENC_API_STRUCT_VERSION(NV_ENC_RC_PARAMS, 1)`

macro for constructing the version field of [\\_NV\\_ENC\\_RC\\_PARAMS](#)

**4.1.1.15** `#define NV_ENC_RECONFIGURE_PARAMS_VER (NVENC_API_STRUCT_VERSION(NV_ENC_RECONFIGURE_PARAMS, 1) | (1 < 31))`

macro for constructing the version field of [\\_NV\\_ENC\\_RECONFIGURE\\_PARAMS](#)

**4.1.1.16** `#define NV_ENC_REGISTER_RESOURCE_VER NVENC_API_STRUCT_VERSION(NV_ENC_REGISTER_RESOURCE, 3)`

Macro for constructing the version field of [\\_NV\\_ENC\\_REGISTER\\_RESOURCE](#)

**4.1.1.17** `#define NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER NVENC_API_STRUCT_VERSION(NV_ENC_SEQUENCE_PARAM_PAYLOAD, 1)`

Macro for constructing the version field of [\\_NV\\_ENC\\_SEQUENCE\\_PARAM\\_PAYLOAD](#)

**4.1.1.18** `#define NV_ENC_STAT_VER NVENC_API_STRUCT_VERSION(NV_ENC_STAT, 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\_PL*** Semi-Planar YUV [UV interleaved] allocated as serial 2D buffer.

***NV\_ENC\_BUFFER\_FORMAT\_NV12\_TILED16x16*** Semi-Planar YUV [UV interleaved] allocated as 16x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_NV12\_TILED64x16*** Semi-Planar YUV [UV interleaved] allocated as 64x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_YV12\_PL*** Planar YUV [YUV separate planes] allocated as serial 2D buffer.

***NV\_ENC\_BUFFER\_FORMAT\_YV12\_TILED16x16*** Planar YUV [YUV separate planes] allocated as 16x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_YV12\_TILED64x16*** Planar YUV [YUV separate planes] allocated as 64x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_IYUV\_PL*** Packed YUV [YUV separate bytes per pixel] allocated as serial 2D buffer.

***NV\_ENC\_BUFFER\_FORMAT\_IYUV\_TILED16x16*** Packed YUV [YUV separate bytes per pixel] allocated as 16x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_IYUV\_TILED64x16*** Packed YUV [YUV separate bytes per pixel] allocated as 64x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_YUV444\_PL*** Planar YUV [YUV separate bytes per pixel] allocated as serial 2D buffer.

***NV\_ENC\_BUFFER\_FORMAT\_YUV444\_TILED16x16*** Planar YUV [YUV separate bytes per pixel] allocated as 16x16 tiles.

***NV\_ENC\_BUFFER\_FORMAT\_YUV444\_TILED64x16*** Planar YUV [YUV separate bytes per pixel] allocated as 64x16 tiles.

#### 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\_RC\_MODE\_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 initialisation. 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 Bufer 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\_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\_CU\_SIZE

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\_SPSPS*** 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 NV\_ENC\_STATUS**

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

#### 4.2.1.4 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](#)

#### 4.2.1.5 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](#)

#### 4.2.1.6 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](#)

4.2.1.7 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](#)

4.2.1.8 NVENCSTATUS NVENCAPI NvEncGetEncodeGUIDCount (void * *encoder*, uint32_t * *encodeGUIDCount*)

The function returns the number of codec guides 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](#)

4.2.1.9 NVENCSTATUS NVENCAPI NvEncGetEncodeGUIDs (void * *encoder*, GUID * *GUIDs*, uint32_t *guidArraySize*, uint32_t * *GUIDCount*)

The function returns an array of codec guides supported by the NvEncodeAPI interface. The client must allocate an array where the NvEncodeAPI interface can fill the supported guides 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 guides 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](#)

4.2.1.10 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 guids 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](#)

4.2.1.11 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](#)

4.2.1.12 NVENCSTATUS NVENCAPI 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](#)

4.2.1.13 NVENCSTATUS NVENCAPI 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 guides 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](#)

4.2.1.14 NVENCSTATUS NVENCAPI NvEncGetEncodeProfileGUIDs (void * *encoder*, GUID *encodeGUID*, GUID * *profileGUIDs*, uint32_t *guidArraySize*, uint32_t * *GUIDCount*)

The function returns an array of supported profile guides for a particular codec guid. The client must allocate an array where the NvEncodeAPI interface can populate the profile guides. 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 guides 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](#)

4.2.1.15 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](#)

4.2.1.16 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](#)

4.2.1.17 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](#)

4.2.1.18 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](#)

4.2.1.19 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](#)

4.2.1.20 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](#)

4.2.1.21 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](#)

4.2.1.22 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](#)

4.2.1.23 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](#)

4.2.1.24 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](#)

4.2.1.25 NVENCSTATUS NVENCAPI NvEncOpenEncodeSession (void **device*, uint32_t *deviceType*, void ***encoder*)

Deprecated.

Returns:

[NV_ENC_ERR_INVALID_CALL](#)

4.2.1.26 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 interface ***device** parameter. If the 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](#)

4.2.1.27 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](#)

4.2.1.28 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](#)

4.2.1.29 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](#)

4.2.1.30 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](#)

4.2.1.31 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](#)

4.2.1.32 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

4.2.1.33 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

4.2.1.34 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_H264_SEI_PAYLOAD` Struct Reference

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

5.6.1 Detailed Description

User SEI message

5.7 _NV_ENC_INITIALIZE_PARAMS Struct Reference

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

5.7.1 Detailed Description

Encode Session Initialization parameters.

5.8 `_NV_ENC_LOCK_BITSTREAM` Struct Reference

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

5.8.1 Detailed Description

Bitstream buffer lock parameters.

5.9 _NV_ENC_LOCK_INPUT_BUFFER Struct Reference

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

5.9.1 Detailed Description

Uncompressed Input Buffer lock parameters.

5.10 `_NV_ENC_MAP_INPUT_RESOURCE` Struct Reference

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

5.10.1 Detailed Description

Map an input resource to a Nvidia Encoder Input Buffer

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_SEQUENCE_PARAM_PAYLOAD Struct Reference

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

5.17.1 Detailed Description

Sequence and picture parameters payload.

5.18 _NV_ENC_STAT Struct Reference

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

5.18.1 Detailed Description

Encode Stats structure.

5.19 _NVENC_EXTERNAL_ME_HINT Struct Reference

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

5.19.1 Detailed Description

External Motion Vector hint structure.

5.20 **_NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE** Struct Reference

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

5.20.1 Detailed Description

External motion vector hint counts per block type.

5.21 _NVENC_RECT Struct Reference

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

5.21.1 Detailed Description

Defines a Rectangle. Used in NV_ENC_PREPROCESS_FRAME.

5.22 GUID Struct Reference

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

Data Fields

- uint32_t [Data1](#)
- uint16_t [Data2](#)
- uint16_t [Data3](#)
- uint8_t [Data4](#) [8]

5.22.1 Detailed Description

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

5.22.2 Field Documentation

5.22.2.1 uint32_t GUID::Data1

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

5.22.2.2 uint16_t GUID::Data2

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

5.22.2.3 uint16_t GUID::Data3

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

5.22.2.4 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.

5.23 NV_ENC_CAPS_PARAM Struct Reference

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

Data Fields

- `uint32_t` [version](#)
- `NV_ENC_CAPS` [capsToQuery](#)
- `uint32_t` [reserved](#) [62]

5.23.1 Detailed Description

Input struct for querying Encoding capabilities.

5.23.2 Field Documentation

5.23.2.1 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.

5.23.2.2 `uint32_t` NV_ENC_CAPS_PARAM::reserved[62]

[in]: Reserved and must be set to 0

5.23.2.3 `uint32_t` NV_ENC_CAPS_PARAM::version

[in]: Struct version. Must be set to [NV_ENC_CAPS_PARAM_VER](#)

5.24 NV_ENC_CODEC_PIC_PARAMS Union Reference

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

Data Fields

- NV_ENC_PIC_PARAMS_H264 [h264PicParams](#)
- NV_ENC_PIC_PARAMS_HEVC [hevcPicParams](#)
- uint32_t [reserved](#) [256]

5.24.1 Detailed Description

Codec specific per-picture encoding parameters.

5.24.2 Field Documentation

5.24.2.1 NV_ENC_PIC_PARAMS_H264 NV_ENC_CODEC_PIC_PARAMS::h264PicParams

[in]: H264 encode picture params.

5.24.2.2 NV_ENC_PIC_PARAMS_HEVC NV_ENC_CODEC_PIC_PARAMS::hevcPicParams

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

5.24.2.3 uint32_t NV_ENC_CODEC_PIC_PARAMS::reserved[256]

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

5.25 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]

5.25.1 Detailed Description

Creation parameters for output bitstream buffer.

5.25.2 Field Documentation

5.25.2.1 NV_ENC_OUTPUT_PTR NV_ENC_CREATE_BITSTREAM_BUFFER::bitstreamBuffer

[out]: Pointer to the output bitstream buffer

5.25.2.2 void* NV_ENC_CREATE_BITSTREAM_BUFFER::bitstreamBufferPtr

[out]: Reserved and should not be used

5.25.2.3 NV_ENC_MEMORY_HEAP NV_ENC_CREATE_BITSTREAM_BUFFER::memoryHeap

[in]: Output buffer memory heap

5.25.2.4 uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::reserved

[in]: Reserved and must be set to 0

5.25.2.5 uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::reserved1[58]

[in]: Reserved and should be set to 0

5.25.2.6 void* NV_ENC_CREATE_BITSTREAM_BUFFER::reserved2[64]

[in]: Reserved and should be set to NULL

5.25.2.7 `uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::size`

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

5.25.2.8 `uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::version`

[in]: Struct version. Must be set to [NV_ENC_CREATE_BITSTREAM_BUFFER_VER](#)

5.26 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]

5.26.1 Detailed Description

Creation parameters for input buffer.

5.26.2 Field Documentation

5.26.2.1 NV_ENC_BUFFER_FORMAT NV_ENC_CREATE_INPUT_BUFFER::bufferFmt

[in]: Input buffer format

5.26.2.2 uint32_t NV_ENC_CREATE_INPUT_BUFFER::height

[in]: Input buffer width

5.26.2.3 NV_ENC_INPUT_PTR NV_ENC_CREATE_INPUT_BUFFER::inputBuffer

[out]: Pointer to input buffer

5.26.2.4 NV_ENC_MEMORY_HEAP NV_ENC_CREATE_INPUT_BUFFER::memoryHeap

[in]: Input buffer memory heap

5.26.2.5 void* NV_ENC_CREATE_INPUT_BUFFER::pSysMemBuffer

[in]: Pointer to existing system buffer

5.26.2.6 uint32_t NV_ENC_CREATE_INPUT_BUFFER::reserved

[in]: Reserved and must be set to 0

5.26.2.7 uint32_t NV_ENC_CREATE_INPUT_BUFFER::reserved1[57]

[in]: Reserved and must be set to 0

5.26.2.8 void* NV_ENC_CREATE_INPUT_BUFFER::reserved2[63]

[in]: Reserved and must be set to NULL

5.26.2.9 uint32_t NV_ENC_CREATE_INPUT_BUFFER::version

[in]: Struct version. Must be set to [NV_ENC_CREATE_INPUT_BUFFER_VER](#)

5.26.2.10 uint32_t NV_ENC_CREATE_INPUT_BUFFER::width

[in]: Input buffer width

5.27 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]

5.27.1 Detailed Description

Event registration/unregistration parameters.

5.27.2 Field Documentation

5.27.2.1 void* NV_ENC_EVENT_PARAMS::completionEvent

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

5.27.2.2 uint32_t NV_ENC_EVENT_PARAMS::reserved

[in]: Reserved and must be set to 0

5.27.2.3 uint32_t NV_ENC_EVENT_PARAMS::reserved1[253]

[in]: Reserved and must be set to 0

5.27.2.4 void* NV_ENC_EVENT_PARAMS::reserved2[64]

[in]: Reserved and must be set to NULL

5.27.2.5 uint32_t NV_ENC_EVENT_PARAMS::version

[in]: Struct version. Must be set to [NV_ENC_EVENT_PARAMS_VER](#).

5.28 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]

5.28.1 Detailed Description

Encoder Session Creation parameters

5.28.2 Field Documentation

5.28.2.1 uint32_t NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::apiVersion

[in]: API version. Should be set to NVENCAPI_VERSION.

5.28.2.2 void* NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::device

[in]: Pointer to client device.

5.28.2.3 NV_ENC_DEVICE_TYPE NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::deviceType

[in]: Specified the device Type

5.28.2.4 void* NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved

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

5.28.2.5 uint32_t NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved1[253]

[in]: Reserved and must be set to 0

5.28.2.6 void* NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved2[64]

[in]: Reserved and must be set to NULL

5.28.2.7 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](#).

5.29 NV_ENC_QP Struct Reference

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

5.29.1 Detailed Description

QP value for frames

5.30 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]

5.30.1 Detailed Description

Rate Control Configuration Paramters

5.30.2 Field Documentation

5.30.2.1 [uint32_t](#) NV_ENC_RC_PARAMS::averageBitRate

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

5.30.2.2 [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.

5.30.2.3 [uint32_t](#) NV_ENC_RC_PARAMS::enableAQ

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

5.30.2.4 [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

5.30.2.5 uint32_t NV_ENC_RC_PARAMS::enableInitialRCQP

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

5.30.2.6 uint32_t NV_ENC_RC_PARAMS::enableMaxQP

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

5.30.2.7 uint32_t NV_ENC_RC_PARAMS::enableMinQP

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

5.30.2.8 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.

5.30.2.9 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.

5.30.2.10 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.

5.30.2.11 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.

5.30.2.12 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.

5.30.2.13 uint32_t NV_ENC_RC_PARAMS::reservedBitFields

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

5.30.2.14 uint32_t NV_ENC_RC_PARAMS::temporalLayerIdxMask

[in]: Specifies the temporal layers (as a bitmask) whose QPs have changed. Valid max bitmask is $[2^{\text{NV_ENC_CAPS_NUM_MAX_TEMPORAL_LAYERS}} - 1]$

5.30.2.15 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

5.30.2.16 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.

5.30.2.17 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 .

5.31 NV_ENCODE_API_FUNCTION_LIST Struct Reference

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

Data Fields

- uint32_t [version](#)
- uint32_t [reserved](#)
- PNVENCODESESSION [nvEncOpenEncodeSession](#)
- PNVEGETENCODEGUIDCOUNT [nvEncGetEncodeGUIDCount](#)
- PNVEGETENCODEPRESETCOUNT [nvEncGetEncodeProfileGUIDCount](#)
- PNVEGETENCODEPRESETGUIDS [nvEncGetEncodeProfileGUIDs](#)
- PNVEGETENCODEGUIDS [nvEncGetEncodeGUIDs](#)
- PNVEGETINPUTFORMATCOUNT [nvEncGetInputFormatCount](#)
- PNVEGETINPUTFORMATS [nvEncGetInputFormats](#)
- PNVEGETENCODECAPS [nvEncGetEncodeCaps](#)
- PNVEGETENCODEPRESETCOUNT [nvEncGetEncodePresetCount](#)
- PNVEGETENCODEPRESETGUIDS [nvEncGetEncodePresetGUIDs](#)
- PNVEGETENCODEPRESETCONFIG [nvEncGetEncodePresetConfig](#)
- PNVEINITIALIZEENCODER [nvEncInitializeEncoder](#)
- PNVECCREATEINPUTBUFFER [nvEncCreateInputBuffer](#)
- PNVECDestroyINPUTBUFFER [nvEncDestroyInputBuffer](#)
- PNVECCREATEBITSTREAMBUFFER [nvEncCreateBitstreamBuffer](#)
- PNVECDestroyBITSTREAMBUFFER [nvEncDestroyBitstreamBuffer](#)
- PNVEENCODEPICTURE [nvEncEncodePicture](#)
- PNVELOCKBITSTREAM [nvEncLockBitstream](#)
- PNVEUNLOCKBITSTREAM [nvEncUnlockBitstream](#)
- PNVELOCKINPUTBUFFER [nvEncLockInputBuffer](#)
- PNVEUNLOCKINPUTBUFFER [nvEncUnlockInputBuffer](#)
- PNVEGETENCODESTATS [nvEncGetEncodeStats](#)
- PNVEGETSEQUENCEPARAMS [nvEncGetSequenceParams](#)
- PNVEREGISTERASYNCEVENT [nvEncRegisterAsyncEvent](#)
- PNVEUNREGISTERASYNCEVENT [nvEncUnregisterAsyncEvent](#)
- PNVECMAPINPUTRESOURCE [nvEncMapInputResource](#)
- PNVEUNMAPINPUTRESOURCE [nvEncUnmapInputResource](#)
- PNVECDestroyENCODER [nvEncDestroyEncoder](#)
- PNVEINVALIDATEREFFRAMES [nvEncInvalidateRefFrames](#)
- PNVEOPENENCODESESSIONEX [nvEncOpenEncodeSessionEx](#)
- PNVEREGISTERRESOURCE [nvEncRegisterResource](#)
- PNVEUNREGISTERRESOURCE [nvEncUnregisterResource](#)
- PNVECRECONFIGUREENCODER [nvEncReconfigureEncoder](#)
- void * [reserved2](#) [285]

5.31.1 Detailed Description

[NV_ENCODE_API_FUNCTION_LIST](#)

5.31.2 Field Documentation

5.31.2.1 PNVENCCREATEBITSTREAMBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncCreateBitstreamBuffer

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

5.31.2.2 PNVENCCREATEINPUTBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncCreateInputBuffer

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

5.31.2.3 PNVENCDESTROYBITSTREAMBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncDestroyBitstreamBuffer

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

5.31.2.4 PNVENCDESTROYENCODER NV_ENCODE_API_FUNCTION_LIST::nvEncDestroyEncoder

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

5.31.2.5 PNVENCDESTROYINPUTBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncDestroyInputBuffer

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

5.31.2.6 PNVENCENCODEPICTURE NV_ENCODE_API_FUNCTION_LIST::nvEncEncodePicture

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

5.31.2.7 PNVENCGETENCODECAPS NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeCaps

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

5.31.2.8 PNVENCGETENCODEGUIDCOUNT NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeGUIDCount

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

5.31.2.9 PNVENCGETENCODEGUIDS NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeGUIDs

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

5.31.2.10 PNVENCGETENCODEPRESETCONFIG NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodePresetConfig

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

5.31.2.11 PNVENCGETENCODEPRESETCOUNT NV_ENCODE_API_FUNCTION_- LIST::nvEncGetEncodePresetCount

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

5.31.2.12 PNVENCGETENCODEPRESETGUIDS NV_ENCODE_API_FUNCTION_- LIST::nvEncGetEncodePresetGUIDs

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

5.31.2.13 PNVENCGETENCODEPRESETCOUNT NV_ENCODE_API_FUNCTION_- LIST::nvEncGetEncodeProfileGUIDCount

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

5.31.2.14 PNVENCGETENCODEPRESETGUIDS NV_ENCODE_API_FUNCTION_- LIST::nvEncGetEncodeProfileGUIDs

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

5.31.2.15 PNVENCGETENCODESTATS NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeStats

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

5.31.2.16 PNVENCGETINPUTFORMATCOUNT NV_ENCODE_API_FUNCTION_- LIST::nvEncGetInputFormatCount

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

5.31.2.17 PNVENCGETINPUTFORMATS NV_ENCODE_API_FUNCTION_- LIST::nvEncGetInputFormats

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

5.31.2.18 PNVENCGETSEQUENCEPARAMS NV_ENCODE_API_FUNCTION_- LIST::nvEncGetSequenceParams

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

5.31.2.19 PNVENCINITIALIZEENCODER NV_ENCODE_API_FUNCTION_- LIST::nvEncInitializeEncoder

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

5.31.2.20 PNVENCINVALIDATEREFFRAMES NV_ENCODE_API_FUNCTION_- LIST::nvEncInvalidateRefFrames

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

5.31.2.21 PNVELOCKBITSTREAM NV_ENCODE_API_FUNCTION_LIST::nvEncLockBitstream

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

5.31.2.22 PNVELOCKINPUTBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncLockInputBuffer

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

5.31.2.23 PNVECMAPINPUTRESOURCE NV_ENCODE_API_FUNCTION_LIST::nvEncMapInputResource

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

5.31.2.24 PNVEOPENENCODESESSION NV_ENCODE_API_FUNCTION_LIST::nvEncOpenEncodeSession

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

5.31.2.25 PNVEOPENENCODESESSIONEX NV_ENCODE_API_FUNCTION_LIST::nvEncOpenEncodeSessionEx

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

5.31.2.26 PNVECRECONFIGUREENCODER NV_ENCODE_API_FUNCTION_LIST::nvEncReconfigureEncoder

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

5.31.2.27 PNVECREGISTERASYNCEVENT NV_ENCODE_API_FUNCTION_LIST::nvEncRegisterAsyncEvent

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

5.31.2.28 PNVECREGISTERRESOURCE NV_ENCODE_API_FUNCTION_LIST::nvEncRegisterResource

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

5.31.2.29 PNVEUNLOCKBITSTREAM NV_ENCODE_API_FUNCTION_LIST::nvEncUnlockBitstream

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

5.31.2.30 PNVEUNLOCKINPUTBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncUnlockInputBuffer

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

5.31.2.31 PNVENCUNMAPINPUTRESOURCE NV_ENCODE_API_FUNCTION_LIST::nvEncUnmapInputResource

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

5.31.2.32 PNVENCUNREGISTERASYNCCEVENT NV_ENCODE_API_FUNCTION_LIST::nvEncUnregisterAsyncEvent

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

5.31.2.33 PNVENCUNREGISTERRESOURCE NV_ENCODE_API_FUNCTION_LIST::nvEncUnregisterResource

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

5.31.2.34 uint32_t NV_ENCODE_API_FUNCTION_LIST::reserved

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

5.31.2.35 void* NV_ENCODE_API_FUNCTION_LIST::reserved2[285]

[in]: Reserved and must be set to NULL

5.31.2.36 uint32_t NV_ENCODE_API_FUNCTION_LIST::version

[in]: Client should pass NV_ENCODE_API_FUNCTION_LIST_VER.

Index

- [_NVENC_EXTERNAL_ME_HINT](#), [63](#)
- [_NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE](#), [64](#)
- [_NVENC_RECT](#), [65](#)
- [_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_H264_SEI_PAYLOAD](#), [50](#)
- [_NV_ENC_INITIALIZE_PARAMS](#), [51](#)
- [_NV_ENC_LOCK_BITSTREAM](#), [52](#)
- [_NV_ENC_LOCK_INPUT_BUFFER](#), [53](#)
- [_NV_ENC_MAP_INPUT_RESOURCE](#), [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_SEQUENCE_PARAM_PAYLOAD](#), [61](#)
- [_NV_ENC_STAT](#), [62](#)
- [apiVersion](#)
 - [NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS](#), [74](#)
- [averageBitRate](#)
 - [NV_ENC_RC_PARAMS](#), [77](#)
- [bitstreamBuffer](#)
 - [NV_ENC_CREATE_BITSTREAM_BUFFER](#), [69](#)
- [bitstreamBufferPtr](#)
 - [NV_ENC_CREATE_BITSTREAM_BUFFER](#), [69](#)
- [bufferFmt](#)
 - [NV_ENC_CREATE_INPUT_BUFFER](#), [71](#)
- [capsToQuery](#)
 - [NV_ENC_CAPS_PARAM](#), [67](#)
- [completionEvent](#)
 - [NV_ENC_EVENT_PARAMS](#), [73](#)
- [constQP](#)
 - [NV_ENC_RC_PARAMS](#), [77](#)
- [Data1](#)
 - [GUID](#), [66](#)
- [Data2](#)
 - [GUID](#), [66](#)
- [Data3](#)
 - [GUID](#), [66](#)
- [Data4](#)
 - [GUID](#), [66](#)
- [device](#)
 - [NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS](#), [74](#)
- [deviceType](#)
 - [NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS](#), [74](#)
- [enableAQ](#)
 - [NV_ENC_RC_PARAMS](#), [77](#)
- [enableExtQPDeltaMap](#)
 - [NV_ENC_RC_PARAMS](#), [77](#)
- [enableInitialRCQP](#)
 - [NV_ENC_RC_PARAMS](#), [77](#)
- [enableMaxQP](#)
 - [NV_ENC_RC_PARAMS](#), [78](#)
- [enableMinQP](#)
 - [NV_ENC_RC_PARAMS](#), [78](#)
- [ENCODE_FUNC](#)
 - [NvEncCreateBitstreamBuffer](#), [23](#)
 - [NvEncCreateInputBuffer](#), [24](#)
 - [NvEncDestroyBitstreamBuffer](#), [24](#)
 - [NvEncDestroyEncoder](#), [25](#)
 - [NvEncDestroyInputBuffer](#), [25](#)
 - [NvEncEncodePicture](#), [26](#)
 - [NvEncGetEncodeCaps](#), [28](#)
 - [NvEncGetEncodeGUIDCount](#), [29](#)
 - [NvEncGetEncodeGUIDs](#), [29](#)
 - [NvEncGetEncodePresetConfig](#), [30](#)
 - [NvEncGetEncodePresetCount](#), [30](#)
 - [NvEncGetEncodePresetGUIDs](#), [31](#)
 - [NvEncGetEncodeProfileGUIDCount](#), [31](#)
 - [NvEncGetEncodeProfileGUIDs](#), [32](#)
 - [NvEncGetEncodeStats](#), [32](#)
 - [NvEncGetInputFormatCount](#), [33](#)
 - [NvEncGetInputFormats](#), [33](#)
 - [NvEncGetSequenceParams](#), [34](#)
 - [NvEncInitializeEncoder](#), [34](#)
 - [NvEncInvalidateRefFrames](#), [36](#)
 - [NvEncLockBitstream](#), [36](#)
 - [NvEncLockInputBuffer](#), [37](#)

- NvEncMapInputResource, 37
- NvEncodeAPICreateInstance, 38
- NvEncOpenEncodeSession, 38
- NvEncOpenEncodeSessionEx, 38
- NvEncReconfigureEncoder, 39
- NvEncRegisterAsyncEvent, 39
- NvEncRegisterResource, 40
- NvEncUnlockBitstream, 40
- NvEncUnlockInputBuffer, 41
- NvEncUnmapInputResource, 41
- NvEncUnregisterAsyncEvent, 42
- NvEncUnregisterResource, 42
- ENCODER_STRUCTURE
 - NV_ENC_BUFFER_FORMAT_IYUV_PL, 12
 - NV_ENC_BUFFER_FORMAT_IYUV_-TILED16x16, 13
 - NV_ENC_BUFFER_FORMAT_IYUV_-TILED64x16, 13
 - NV_ENC_BUFFER_FORMAT_NV12_PL, 12
 - NV_ENC_BUFFER_FORMAT_NV12_-TILED16x16, 12
 - NV_ENC_BUFFER_FORMAT_NV12_-TILED64x16, 12
 - NV_ENC_BUFFER_FORMAT_UNDEFINED, 12
 - NV_ENC_BUFFER_FORMAT_YUV444_PL, 13
 - NV_ENC_BUFFER_FORMAT_YUV444_-TILED16x16, 13
 - NV_ENC_BUFFER_FORMAT_YUV444_-TILED64x16, 13
 - NV_ENC_BUFFER_FORMAT_YV12_PL, 12
 - NV_ENC_BUFFER_FORMAT_YV12_-TILED16x16, 12
 - NV_ENC_BUFFER_FORMAT_YV12_-TILED64x16, 12
 - NV_ENC_CAPS_ASYNC_ENCODE_SUPPORT, 15
 - NV_ENC_CAPS_EXPOSED_COUNT, 15
 - NV_ENC_CAPS_HEIGHT_MAX, 14
 - NV_ENC_CAPS_LEVEL_MAX, 14
 - NV_ENC_CAPS_LEVEL_MIN, 14
 - NV_ENC_CAPS_MB_NUM_MAX, 15
 - NV_ENC_CAPS_MB_PER_SEC_MAX, 15
 - NV_ENC_CAPS_NUM_MAX_BFRAMES, 13
 - NV_ENC_CAPS_NUM_MAX_TEMPORAL_-LAYERS, 13
 - NV_ENC_CAPS_PREPROC_SUPPORT, 15
 - NV_ENC_CAPS_SEPARATE_COLOUR_PLANE, 14
 - 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, 15
 - NV_ENC_CAPS_SUPPORT_DYN_BITRATE_-CHANGE, 14
 - NV_ENC_CAPS_SUPPORT_DYN_FORCE_-CONSTQP, 14
 - NV_ENC_CAPS_SUPPORT_DYN_RC_MODE_-CHANGE, 14
 - NV_ENC_CAPS_SUPPORT_DYN_RES_-CHANGE, 14
 - NV_ENC_CAPS_SUPPORT_DYNAMIC_-SLICE_MODE, 15
 - NV_ENC_CAPS_SUPPORT_FIELD_-ENCODING, 13
 - NV_ENC_CAPS_SUPPORT_FMO, 13
 - NV_ENC_CAPS_SUPPORT_HIERARCHICAL_-BFRAMES, 14
 - NV_ENC_CAPS_SUPPORT_HIERARCHICAL_-PFRAMES, 14
 - NV_ENC_CAPS_SUPPORT_INTRA_REFRESH, 14
 - NV_ENC_CAPS_SUPPORT_LOSSLESS_-ENCODE, 15
 - NV_ENC_CAPS_SUPPORT_MONOCHROME, 13
 - NV_ENC_CAPS_SUPPORT_QPELMV, 13
 - NV_ENC_CAPS_SUPPORT_REF_PIC_-INVALIDATION, 15
 - NV_ENC_CAPS_SUPPORT_RESERVED, 13
 - NV_ENC_CAPS_SUPPORT_SUBFRAME_-READBACK, 14
 - NV_ENC_CAPS_SUPPORT_TEMPORAL_SVC, 14
 - NV_ENC_CAPS_SUPPORT_YUV444_ENCODE, 15
 - NV_ENC_CAPS_SUPPORTED_-RATECONTROL_MODES, 13
 - NV_ENC_CAPS_WIDTH_MAX, 14
 - NV_ENC_DEVICE_TYPE_CUDA, 15
 - NV_ENC_DEVICE_TYPE_DIRECTX, 15
 - NV_ENC_ERR_DEVICE_NOT_EXIST, 19
 - NV_ENC_ERR_ENCODER_BUSY, 20
 - NV_ENC_ERR_ENCODER_NOT_INITIALIZED, 19
 - NV_ENC_ERR_EVENT_NOT_REGISTERD, 20
 - NV_ENC_ERR_GENERIC, 20
 - NV_ENC_ERR_INCOMPATIBLE_CLIENT_-KEY, 20
 - 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, 20
 NV_ENC_ERR_LOCK_BUSY, 19
 NV_ENC_ERR_MAP_FAILED, 20
 NV_ENC_ERR_NEED_MORE_INPUT, 20
 NV_ENC_ERR_NO_ENCODE_DEVICE, 19
 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, 20
 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, 16
 NV_ENC_H264_BDIRECT_MODE_DISABLE, 16
 NV_ENC_H264_BDIRECT_MODE_SPATIAL, 16
 NV_ENC_H264_BDIRECT_MODE_TEMPORAL, 16
 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, 17
 NV_ENC_MEMORY_HEAP_SYSMEM_CACHED, 17
 NV_ENC_MEMORY_HEAP_SYSMEM_UNCACHED, 17
 NV_ENC_MEMORY_HEAP_VID, 17
 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_FRAME_SIZE_CAP, 18
 NV_ENC_PARAMS_RC_2_PASS_QUALITY, 17
 NV_ENC_PARAMS_RC_2_PASS_VBR, 18
 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, 18
 NV_ENC_PIC_FLAG_FORCEIDR, 18
 NV_ENC_PIC_FLAG_FORCEINTRA, 18
 NV_ENC_PIC_FLAG_OUTPUT_SPSPPS, 18
 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, 19
 NV_ENC_STEREO_PACKING_MODE_COLINTERLEAVE, 19
 NV_ENC_STEREO_PACKING_MODE_FRAMESEQ, 19
 NV_ENC_STEREO_PACKING_MODE_NONE, 19
 NV_ENC_STEREO_PACKING_MODE_ROWINTERLEAVE, 19
 NV_ENC_STEREO_PACKING_MODE_SIDE_BY_SIDE, 19
 NV_ENC_STEREO_PACKING_MODE_TOPBOTTOM, 19
 NV_ENC_SUCCESS, 19
 ENCODER_STRUCTURE
 NV_ENC_BUFFER_FORMAT, 12
 NV_ENC_CAPS, 13
 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_DEVICE_TYPE, 15
- NV_ENC_EVENT_PARAMS_VER, 11
- NV_ENC_H264_ADAPTIVE_TRANSFORM_MODE, 15
- NV_ENC_H264_BDIRECT_MODE, 15
- NV_ENC_H264_ENTROPY_CODING_MODE, 16
- 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, 17
- NV_ENC_MV_PRECISION, 17
- 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, 18
- NV_ENC_PIC_PARAMS_VER, 11
- NV_ENC_PIC_STRUCT, 18
- NV_ENC_PIC_TYPE, 18
- NV_ENC_PRESET_CONFIG_VER, 11
- NV_ENC_RC_PARAMS_VER, 12
- NV_ENC_RECONFIGURE_PARAMS_VER, 12
- 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, 19
- GUID, 66
 - Data1, 66
 - Data2, 66
 - Data3, 66
 - Data4, 66
- h264PicParams
 - NV_ENC_CODEC_PIC_PARAMS, 68
- height
 - NV_ENC_CREATE_INPUT_BUFFER, 71
- hevcPicParams
 - NV_ENC_CODEC_PIC_PARAMS, 68
- initialRCQP
 - NV_ENC_RC_PARAMS, 78
- inputBuffer
 - NV_ENC_CREATE_INPUT_BUFFER, 71
- maxBitRate
 - NV_ENC_RC_PARAMS, 78
- maxQP
 - NV_ENC_RC_PARAMS, 78
- memoryHeap
 - NV_ENC_CREATE_BITSTREAM_BUFFER, 69
 - NV_ENC_CREATE_INPUT_BUFFER, 71
- minQP
 - NV_ENC_RC_PARAMS, 78
- NV_ENC_BUFFER_FORMAT_IYUV_PL
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_IYUV_TILED16x16
 - ENCODER_STRUCTURE, 13
- NV_ENC_BUFFER_FORMAT_IYUV_TILED64x16
 - ENCODER_STRUCTURE, 13
- NV_ENC_BUFFER_FORMAT_NV12_PL
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_NV12_TILED16x16
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_NV12_TILED64x16
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_UNDEFINED
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_YUV444_PL
 - ENCODER_STRUCTURE, 13
- NV_ENC_BUFFER_FORMAT_YUV444_TILED16x16
 - ENCODER_STRUCTURE, 13
- NV_ENC_BUFFER_FORMAT_YUV444_TILED64x16
 - ENCODER_STRUCTURE, 13
- NV_ENC_BUFFER_FORMAT_YV12_PL
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_YV12_TILED16x16
 - ENCODER_STRUCTURE, 12
- NV_ENC_BUFFER_FORMAT_YV12_TILED64x16
 - ENCODER_STRUCTURE, 12
- NV_ENC_CAPS_ASYNC_ENCODE_SUPPORT
 - ENCODER_STRUCTURE, 15
- NV_ENC_CAPS_EXPOSED_COUNT
 - ENCODER_STRUCTURE, 15
- NV_ENC_CAPS_HEIGHT_MAX
 - ENCODER_STRUCTURE, 14
- NV_ENC_CAPS_LEVEL_MAX
 - ENCODER_STRUCTURE, 14
- NV_ENC_CAPS_LEVEL_MIN
 - ENCODER_STRUCTURE, 14
- NV_ENC_CAPS_MB_NUM_MAX
 - ENCODER_STRUCTURE, 15
- NV_ENC_CAPS_MB_PER_SEC_MAX
 - ENCODER_STRUCTURE, 15
- NV_ENC_CAPS_NUM_MAX_BFRAMES
 - ENCODER_STRUCTURE, 13

- NV_ENC_CAPS_NUM_MAX_TEMPORAL_LAYERS
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_PREPROC_SUPPORT
ENCODER_STRUCTURE, [15](#)
- NV_ENC_CAPS_SEPARATE_COLOUR_PLANE
ENCODER_STRUCTURE, [14](#)
- 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, [15](#)
- NV_ENC_CAPS_SUPPORT_DYN_BITRATE_-
CHANGE
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_DYN_FORCE_-
CONSTQP
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_DYN_RC_MODE_-
CHANGE
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_DYN_RES_CHANGE
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_DYNAMIC_SLICE_-
MODE
ENCODER_STRUCTURE, [15](#)
- NV_ENC_CAPS_SUPPORT_FIELD_ENCODING
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_SUPPORT_FMO
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_SUPPORT_HIERARCHICAL_-
BFRAMES
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_HIERARCHICAL_-
PFRAMES
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_INTRA_REFRESH
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_LOSSLESS_ENCODE
ENCODER_STRUCTURE, [15](#)
- NV_ENC_CAPS_SUPPORT_MONOCHROME
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_SUPPORT_QPELMV
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_SUPPORT_REF_PIC_-
INVALIDATION
ENCODER_STRUCTURE, [15](#)
- NV_ENC_CAPS_SUPPORT_RESERVED
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_SUPPORT_SUBFRAME_-
READBACK
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_TEMPORAL_SVC
ENCODER_STRUCTURE, [14](#)
- NV_ENC_CAPS_SUPPORT_YUV444_ENCODE
ENCODER_STRUCTURE, [15](#)
- NV_ENC_CAPS_SUPPORTED_RATECONTROL_-
MODES
ENCODER_STRUCTURE, [13](#)
- NV_ENC_CAPS_WIDTH_MAX
ENCODER_STRUCTURE, [14](#)
- 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, [20](#)
- NV_ENC_ERR_ENCODER_NOT_INITIALIZED
ENCODER_STRUCTURE, [19](#)
- NV_ENC_ERR_EVENT_NOT_REGISTERD
ENCODER_STRUCTURE, [20](#)
- NV_ENC_ERR_GENERIC
ENCODER_STRUCTURE, [20](#)
- NV_ENC_ERR_INCOMPATIBLE_CLIENT_KEY
ENCODER_STRUCTURE, [20](#)
- 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, [20](#)
- NV_ENC_ERR_LOCK_BUSY
ENCODER_STRUCTURE, [19](#)
- NV_ENC_ERR_MAP_FAILED
ENCODER_STRUCTURE, [20](#)
- NV_ENC_ERR_NEED_MORE_INPUT
ENCODER_STRUCTURE, [20](#)
- NV_ENC_ERR_NO_ENCODE_DEVICE
ENCODER_STRUCTURE, [19](#)
- 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, 20
- 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, 16
- NV_ENC_H264_BDIRECT_MODE_DISABLE
ENCODER_STRUCTURE, 16
- NV_ENC_H264_BDIRECT_MODE_SPATIAL
ENCODER_STRUCTURE, 16
- NV_ENC_H264_BDIRECT_MODE_TEMPORAL
ENCODER_STRUCTURE, 16
- 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, 17
- NV_ENC_MEMORY_HEAP_SYMMEM_CACHED
ENCODER_STRUCTURE, 17
- NV_ENC_MEMORY_HEAP_SYMMEM_UNCACHED
ENCODER_STRUCTURE, 17
- NV_ENC_MEMORY_HEAP_VID
ENCODER_STRUCTURE, 17
- 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, 18
- NV_ENC_PARAMS_RC_2_PASS_QUALITY
ENCODER_STRUCTURE, 17
- NV_ENC_PARAMS_RC_2_PASS_VBR
ENCODER_STRUCTURE, 18
- 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, 18
- NV_ENC_PIC_FLAG_FORCEIDR
ENCODER_STRUCTURE, 18
- NV_ENC_PIC_FLAG_FORCEINTRA
ENCODER_STRUCTURE, 18
- NV_ENC_PIC_FLAG_OUTPUT_SPSPPS
ENCODER_STRUCTURE, 18
- 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, 19
- NV_ENC_STEREO_PACKING_MODE_-COLINTERLEAVE
 - ENCODER_STRUCTURE, 19
- NV_ENC_STEREO_PACKING_MODE_FRAMESEQ
 - ENCODER_STRUCTURE, 19
- NV_ENC_STEREO_PACKING_MODE_NONE
 - ENCODER_STRUCTURE, 19
- NV_ENC_STEREO_PACKING_MODE_-ROWINTERLEAVE
 - ENCODER_STRUCTURE, 19
- NV_ENC_STEREO_PACKING_MODE_SIDE BYSIDE
 - ENCODER_STRUCTURE, 19
- NV_ENC_STEREO_PACKING_MODE_-TOPBOTTOM
 - ENCODER_STRUCTURE, 19
- NV_ENC_SUCCESS
 - ENCODER_STRUCTURE, 19
- NV_ENC_BUFFER_FORMAT
 - ENCODER_STRUCTURE, 12
- NV_ENC_CAPS
 - ENCODER_STRUCTURE, 13
- NV_ENC_CAPS_PARAM, 67
 - capsToQuery, 67
 - reserved, 67
 - version, 67
- NV_ENC_CAPS_PARAM_VER
 - ENCODER_STRUCTURE, 10
- NV_ENC_CODEC_PIC_PARAMS, 68
 - h264PicParams, 68
 - hevcPicParams, 68
 - reserved, 68
- NV_ENC_CONFIG_VER
 - ENCODER_STRUCTURE, 10
- NV_ENC_CREATE_BITSTREAM_BUFFER, 69
 - bitstreamBuffer, 69
 - bitstreamBufferPtr, 69
 - memoryHeap, 69
 - reserved, 69
 - reserved1, 69
 - reserved2, 69
 - size, 69
 - version, 70
- NV_ENC_CREATE_BITSTREAM_BUFFER_VER
 - ENCODER_STRUCTURE, 10
- NV_ENC_CREATE_INPUT_BUFFER, 71
 - bufferFmt, 71
 - height, 71
 - inputBuffer, 71
 - memoryHeap, 71
 - pSysMemBuffer, 71
 - reserved, 71
 - reserved1, 71
 - reserved2, 72
 - version, 72
 - width, 72
- NV_ENC_CREATE_INPUT_BUFFER_VER
 - ENCODER_STRUCTURE, 10
- NV_ENC_DEVICE_TYPE
 - ENCODER_STRUCTURE, 15
- NV_ENC_EVENT_PARAMS, 73
 - completionEvent, 73
 - reserved, 73
 - reserved1, 73
 - reserved2, 73
 - version, 73
- NV_ENC_EVENT_PARAMS_VER
 - ENCODER_STRUCTURE, 11
- 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, 16
- NV_ENC_H264_FMO_MODE
 - ENCODER_STRUCTURE, 16
- 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, 17
- NV_ENC_MV_PRECISION

- ENCODER_STRUCTURE, 17
- NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 74
 - apiVersion, 74
 - device, 74
 - deviceType, 74
 - reserved, 74
 - reserved1, 74
 - reserved2, 74
 - version, 74
- 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, 18
- NV_ENC_PIC_PARAMS_VER
 - ENCODER_STRUCTURE, 11
- NV_ENC_PIC_STRUCT
 - ENCODER_STRUCTURE, 18
- NV_ENC_PIC_TYPE
 - ENCODER_STRUCTURE, 18
- NV_ENC_PRESET_CONFIG_VER
 - ENCODER_STRUCTURE, 11
- NV_ENC_QP, 76
- NV_ENC_RC_PARAMS, 77
 - averageBitRate, 77
 - constQP, 77
 - enableAQ, 77
 - enableExtQPDeltaMap, 77
 - enableInitialRCQP, 77
 - enableMaxQP, 78
 - enableMinQP, 78
 - initialRCQP, 78
 - maxBitRate, 78
 - maxQP, 78
 - minQP, 78
 - rateControlMode, 78
 - reservedBitFields, 78
 - temporalLayerIdxMask, 78
 - temporalLayerQP, 78
 - vbvBufferSize, 78
 - vbvInitialDelay, 79
- NV_ENC_RC_PARAMS_VER
 - ENCODER_STRUCTURE, 12
- NV_ENC_RECONFIGURE_PARAMS_VER
 - ENCODER_STRUCTURE, 12
- 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, 80
 - nvEncCreateBitstreamBuffer, 81
 - nvEncCreateInputBuffer, 81
 - nvEncDestroyBitstreamBuffer, 81
 - nvEncDestroyEncoder, 81
 - nvEncDestroyInputBuffer, 81
 - nvEncEncodePicture, 81
 - nvEncGetEncodeCaps, 81
 - nvEncGetEncodeGUIDCount, 81
 - nvEncGetEncodeGUIDs, 81
 - nvEncGetEncodePresetConfig, 81
 - nvEncGetEncodePresetCount, 81
 - nvEncGetEncodePresetGUIDs, 82
 - nvEncGetEncodeProfileGUIDCount, 82
 - nvEncGetEncodeProfileGUIDs, 82
 - nvEncGetEncodeStats, 82
 - nvEncGetInputFormatCount, 82
 - nvEncGetInputFormats, 82
 - nvEncGetSequenceParams, 82
 - nvEncInitializeEncoder, 82
 - nvEncInvalidateRefFrames, 82
 - nvEncLockBitstream, 82
 - nvEncLockInputBuffer, 83
 - nvEncMapInputResource, 83
 - nvEncOpenEncodeSession, 83
 - nvEncOpenEncodeSessionEx, 83
 - nvEncReconfigureEncoder, 83
 - nvEncRegisterAsyncEvent, 83
 - nvEncRegisterResource, 83
 - nvEncUnlockBitstream, 83
 - nvEncUnlockInputBuffer, 83
 - nvEncUnmapInputResource, 83
 - nvEncUnregisterAsyncEvent, 84
 - nvEncUnregisterResource, 84
 - reserved, 84
 - reserved2, 84
 - version, 84
- NvEncCreateBitstreamBuffer
 - ENCODE_FUNC, 23
- nvEncCreateBitstreamBuffer
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncCreateInputBuffer
 - ENCODE_FUNC, 24
- nvEncCreateInputBuffer
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncDestroyBitstreamBuffer
 - ENCODE_FUNC, 24
- nvEncDestroyBitstreamBuffer

- NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncDestroyEncoder
 - ENCODE_FUNC, 25
- nvEncDestroyEncoder
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncDestroyInputBuffer
 - ENCODE_FUNC, 25
- nvEncDestroyInputBuffer
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncEncodePicture
 - ENCODE_FUNC, 26
- nvEncEncodePicture
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncGetEncodeCaps
 - ENCODE_FUNC, 28
- nvEncGetEncodeCaps
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncGetEncodeGUIDCount
 - ENCODE_FUNC, 29
- nvEncGetEncodeGUIDCount
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncGetEncodeGUIDs
 - ENCODE_FUNC, 29
- nvEncGetEncodeGUIDs
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncGetEncodePresetConfig
 - ENCODE_FUNC, 30
- nvEncGetEncodePresetConfig
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncGetEncodePresetCount
 - ENCODE_FUNC, 30
- nvEncGetEncodePresetCount
 - NV_ENCODE_API_FUNCTION_LIST, 81
- NvEncGetEncodePresetGUIDs
 - ENCODE_FUNC, 31
- nvEncGetEncodePresetGUIDs
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncGetEncodeProfileGUIDCount
 - ENCODE_FUNC, 31
- nvEncGetEncodeProfileGUIDCount
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncGetEncodeProfileGUIDs
 - ENCODE_FUNC, 32
- nvEncGetEncodeProfileGUIDs
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncGetEncodeStats
 - ENCODE_FUNC, 32
- nvEncGetEncodeStats
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncGetInputFormatCount
 - ENCODE_FUNC, 33
- nvEncGetInputFormatCount
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncGetInputFormats
 - ENCODE_FUNC, 33
- nvEncGetInputFormats
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncInitializeEncoder
 - ENCODE_FUNC, 34
- nvEncInitializeEncoder
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncInvalidateRefFrames
 - ENCODE_FUNC, 36
- nvEncInvalidateRefFrames
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncLockBitstream
 - ENCODE_FUNC, 36
- nvEncLockBitstream
 - NV_ENCODE_API_FUNCTION_LIST, 82
- NvEncLockInputBuffer
 - ENCODE_FUNC, 37
- nvEncLockInputBuffer
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncMapInputResource
 - ENCODE_FUNC, 37
- nvEncMapInputResource
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncodeAPI Data structures, 7
- NvEncodeAPI Functions, 21
- NvEncodeAPICreateInstance
 - ENCODE_FUNC, 38
- NvEncOpenEncodeSession
 - ENCODE_FUNC, 38
- nvEncOpenEncodeSession
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncOpenEncodeSessionEx
 - ENCODE_FUNC, 38
- nvEncOpenEncodeSessionEx
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncReconfigureEncoder
 - ENCODE_FUNC, 39
- nvEncReconfigureEncoder
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncRegisterAsyncEvent
 - ENCODE_FUNC, 39
- nvEncRegisterAsyncEvent
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncRegisterResource
 - ENCODE_FUNC, 40
- nvEncRegisterResource
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NVENCSTATUS
 - ENCODER_STRUCTURE, 19
- NvEncUnlockBitstream

- ENCODE_FUNC, 40
- nvEncUnlockBitstream
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncUnlockInputBuffer
 - ENCODE_FUNC, 41
- nvEncUnlockInputBuffer
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncUnmapInputResource
 - ENCODE_FUNC, 41
- nvEncUnmapInputResource
 - NV_ENCODE_API_FUNCTION_LIST, 83
- NvEncUnregisterAsyncEvent
 - ENCODE_FUNC, 42
- nvEncUnregisterAsyncEvent
 - NV_ENCODE_API_FUNCTION_LIST, 84
- NvEncUnregisterResource
 - ENCODE_FUNC, 42
- nvEncUnregisterResource
 - NV_ENCODE_API_FUNCTION_LIST, 84
- pSysMemBuffer
 - NV_ENC_CREATE_INPUT_BUFFER, 71
- rateControlMode
 - NV_ENC_RC_PARAMS, 78
- reserved
 - NV_ENC_CAPS_PARAM, 67
 - NV_ENC_CODEC_PIC_PARAMS, 68
 - NV_ENC_CREATE_BITSTREAM_BUFFER, 69
 - NV_ENC_CREATE_INPUT_BUFFER, 71
 - NV_ENC_EVENT_PARAMS, 73
 - NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 74
 - NV_ENCODE_API_FUNCTION_LIST, 84
- reserved1
 - NV_ENC_CREATE_BITSTREAM_BUFFER, 69
 - NV_ENC_CREATE_INPUT_BUFFER, 71
 - NV_ENC_EVENT_PARAMS, 73
 - NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 74
- reserved2
 - NV_ENC_CREATE_BITSTREAM_BUFFER, 69
 - NV_ENC_CREATE_INPUT_BUFFER, 72
 - NV_ENC_EVENT_PARAMS, 73
 - NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 74
 - NV_ENCODE_API_FUNCTION_LIST, 84
- reservedBitFields
 - NV_ENC_RC_PARAMS, 78
- size
 - NV_ENC_CREATE_BITSTREAM_BUFFER, 69
- temporalLayerQP
 - NV_ENC_RC_PARAMS, 78
- vbvBufferSize
 - NV_ENC_RC_PARAMS, 78
- vbvInitialDelay
 - NV_ENC_RC_PARAMS, 79
- version
 - NV_ENC_CAPS_PARAM, 67
 - NV_ENC_CREATE_BITSTREAM_BUFFER, 70
 - NV_ENC_CREATE_INPUT_BUFFER, 72
 - NV_ENC_EVENT_PARAMS, 73
 - NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS, 74
 - NV_ENCODE_API_FUNCTION_LIST, 84
- width
 - NV_ENC_CREATE_INPUT_BUFFER, 72

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.