NVENC - NVIDIA Video Encoder API Reference Manual

June 10, 2016

Version 7.0

Contents

Chapter 1

Legal Notice

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 NvEncodeAPI Data structures

Data Structures

- struct GUID
- struct NV_ENC_CAPS_PARAM
- struct NV_ENC_CREATE_INPUT_BUFFER
- struct NV_ENC_CREATE_BITSTREAM_BUFFER
- struct NV_ENC_MVECTOR
- struct NV_ENC_H264_MV_DATA
- struct NV_ENC_HEVC_MV_DATA
- struct NV_ENC_CREATE_MV_BUFFER
- struct NV_ENC_QP
- struct NV_ENC_RC_PARAMS
- union NV_ENC_CODEC_PIC_PARAMS
- struct NV_ENC_EVENT_PARAMS
- struct NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS
- struct NV_ENCODE_API_FUNCTION_LIST
- struct _NVENC_RECT
- struct _NV_ENC_CONFIG_H264_VUI_PARAMETERS
- struct _NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE
- struct _NVENC_EXTERNAL_ME_HINT
- struct _NV_ENC_CONFIG_H264
- struct _NV_ENC_CONFIG_HEVC
- struct _NV_ENC_CODEC_CONFIG
- struct _NV_ENC_CONFIG
- struct _NV_ENC_INITIALIZE_PARAMS
- struct _NV_ENC_RECONFIGURE_PARAMS
- struct _NV_ENC_PRESET_CONFIG
- struct _NV_ENC_SEI_PAYLOAD
- struct _NV_ENC_PIC_PARAMS_H264
- struct _NV_ENC_PIC_PARAMS_HEVC
- struct _NV_ENC_PIC_PARAMS
- struct _NV_ENC_MEONLY_PARAMS
- struct _NV_ENC_LOCK_BITSTREAM

- struct _NV_ENC_LOCK_INPUT_BUFFER
- struct _NV_ENC_MAP_INPUT_RESOURCE
- struct _NV_ENC_REGISTER_RESOURCE
- struct _NV_ENC_STAT
- struct _NV_ENC_SEQUENCE_PARAM_PAYLOAD

Defines

- #define NV_ENC_PARAMS_RC_CBR2 NV_ENC_PARAMS_RC_CBR
- #define NV_ENC_CAPS_PARAM_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_CREATE_INPUT_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_CREATE_BITSTREAM_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_CREATE_MV_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_RC_PARAMS_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_CONFIG_VER (NVENCAPI_STRUCT_VERSION(6) | (1<<31))
- #define NV_ENC_INITIALIZE_PARAMS_VER (NVENCAPI_STRUCT_VERSION(5) | (1<<31))
- #define NV_ENC_RECONFIGURE_PARAMS_VER (NVENCAPI_STRUCT_VERSION(1) | (1<<31))
- #define NV_ENC_PRESET_CONFIG_VER (NVENCAPI_STRUCT_VERSION(4) | (1<<31))
- #define NV_ENC_PIC_PARAMS_VER (NVENCAPI_STRUCT_VERSION(4) | (1<<31))
- #define NV_ENC_MEONLY_PARAMS_VER NVENCAPI_STRUCT_VERSION(2)
- #define NV_ENC_LOCK_BITSTREAM_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_LOCK_INPUT_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_MAP_INPUT_RESOURCE_VER NVENCAPI_STRUCT_VERSION(4)
- #define NV_ENC_REGISTER_RESOURCE_VER NVENCAPI_STRUCT_VERSION(3)
- #define NV_ENC_STAT_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_EVENT_PARAMS_VER NVENCAPI_STRUCT_VERSION(1)
- #define NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER NVENCAPI_STRUCT_VERSION(1)

Enumerations

- enum NV_ENC_PARAMS_FRAME_FIELD_MODE { NV_ENC_PARAMS_FRAME_FIELD_MODE_-FRAME = 0x01, NV_ENC_PARAMS_FRAME_FIELD_MODE_FIELD = 0x02, NV_ENC_PARAMS_-FRAME_FIELD_MODE_MBAFF = 0x03 }
- enum NV_ENC_PARAMS_RC_MODE {

  NV_ENC_PARAMS_RC_CONSTQP = 0x0, NV_ENC_PARAMS_RC_VBR = 0x1, NV_ENC_PARAMS_-RC_CBR = 0x2, NV_ENC_PARAMS_RC_VBR_MINQP = 0x4,

  NV_ENC_PARAMS_RC_2_PASS_QUALITY = 0x8, NV_ENC_PARAMS_RC_2_PASS_FRAMESIZE_-CAP = 0x10, NV_ENC_PARAMS_RC_2_PASS_VBR = 0x20 }
- enum NV_ENC_PIC_STRUCT { NV_ENC_PIC_STRUCT_FRAME = 0x01, NV_ENC_PIC_STRUCT_-FIELD_TOP_BOTTOM = 0x02, NV_ENC_PIC_STRUCT_FIELD_BOTTOM_TOP = 0x03 }
- enum NV_ENC_PIC_TYPE {

  NV_ENC_PIC_TYPE_P = 0x0, NV_ENC_PIC_TYPE_B = 0x01, NV_ENC_PIC_TYPE_I = 0x02, NV_-ENC_PIC_TYPE_IDR = 0x03,

  NV_ENC_PIC_TYPE_BI = 0x04, NV_ENC_PIC_TYPE_SKIPPED = 0x05, NV_ENC_PIC_TYPE_INTRA_-REFRESH = 0x06, NV_ENC_PIC_TYPE_UNKNOWN = 0xFF }
- enum NV_ENC_MV_PRECISION { NV_ENC_MV_PRECISION_DEFAULT = 0x0, NV_ENC_MV_-PRECISION_FULL_PEL = 0x01, NV_ENC_MV_PRECISION_HALF_PEL = 0x02, NV_ENC_MV_-PRECISION_QUARTER_PEL = 0x03 }

- enum NV_ENC_BUFFER_FORMAT {

  NV_ENC_BUFFER_FORMAT_UNDEFINED = 0x00000000, NV_ENC_BUFFER_FORMAT_NV12 = 0x00000001, NV_ENC_BUFFER_FORMAT_YV12 = 0x00000010, NV_ENC_BUFFER_FORMAT_IYUV = 0x00000100,

  NV_ENC_BUFFER_FORMAT_YUV444 = 0x00001000, NV_ENC_BUFFER_FORMAT_YUV420_10BIT = 0x00010000, NV_ENC_BUFFER_FORMAT_YUV444_10BIT = 0x00100000, NV_ENC_BUFFER_-FORMAT_ARGB = 0x01000000,

  NV_ENC_BUFFER_FORMAT_ARGB10 = 0x02000000, NV_ENC_BUFFER_FORMAT_AYUV = 0x04000000, NV_ENC_BUFFER_FORMAT_ABGR = 0x10000000, NV_ENC_BUFFER_FORMAT_-ABGR10 = 0x20000000 }
- enum NV_ENC_LEVEL
- enum NVENCSTATUS {

  NV_ENC_SUCCESS, NV_ENC_ERR_NO_ENCODE_DEVICE, NV_ENC_ERR_UNSUPPORTED_-DEVICE, NV_ENC_ERR_INVALID_ENCODERDEVICE,

  NV_ENC_ERR_INVALID_DEVICE, NV_ENC_ERR_DEVICE_NOT_EXIST, NV_ENC_ERR_INVALID_-PTR, NV_ENC_ERR_INVALID_EVENT,

  NV_ENC_ERR_INVALID_PARAM, NV_ENC_ERR_INVALID_CALL, NV_ENC_ERR_OUT_OF_-MEMORY, NV_ENC_ERR_ENCODER_NOT_INITIALIZED,

  NV_ENC_ERR_UNSUPPORTED_PARAM, NV_ENC_ERR_LOCK_BUSY, NV_ENC_ERR_NOT_-ENOUGH_BUFFER, NV_ENC_ERR_INVALID_VERSION,

  NV_ENC_ERR_MAP_FAILED, NV_ENC_ERR_NEED_MORE_INPUT, NV_ENC_ERR_ENCODER_-BUSY, NV_ENC_ERR_EVENT_NOT_REGISTERD,

  NV_ENC_ERR_GENERIC, NV_ENC_ERR_INCOMPATIBLE_CLIENT_KEY, NV_ENC_ERR_-UNIMPLEMENTED, NV_ENC_ERR_RESOURCE_REGISTER_FAILED,

  NV_ENC_ERR_RESOURCE_NOT_REGISTERED, NV_ENC_ERR_RESOURCE_NOT_MAPPED }
- enum NV_ENC_PIC_FLAGS { NV_ENC_PIC_FLAG_FORCEINTRA = 0x1, NV_ENC_PIC_FLAG_-FORCEIDR = 0x2, NV_ENC_PIC_FLAG_OUTPUT_SPSPPS = 0x4, NV_ENC_PIC_FLAG_EOS = 0x8 }
- enum NV_ENC_MEMORY_HEAP { NV_ENC_MEMORY_HEAP_AUTOSELECT = 0, NV_ENC_-MEMORY_HEAP_VID = 1, NV_ENC_MEMORY_HEAP_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_SIDEBYSIDE = 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_SUPPORT_SAO,

  NV_ENC_CAPS_SUPPORT_MEONLY_MODE, NV_ENC_CAPS_SUPPORT_LOOKAHEAD, NV_ENC_CAPS_SUPPORT_TEMPORAL_AQ, NV_ENC_CAPS_SUPPORT_10BIT_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(1)

NV_ENC_CAPS_PARAM struct version.

4.1.1.2 #define NV_ENC_CONFIG_VER (NVENCAPI_STRUCT_VERSION(6) | (1<<31))

macro for constructing the version field of _NV_ENC_CONFIG

4.1.1.3 #define NV_ENC_CREATE_BITSTREAM_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)

NV_ENC_CREATE_BITSTREAM_BUFFER struct version.

4.1.1.4 #define NV_ENC_CREATE_INPUT_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)

NV_ENC_CREATE_INPUT_BUFFER struct version.

4.1.1.5 #define NV_ENC_CREATE_MV_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)

NV_ENC_CREATE_MV_BUFFER struct version

4.1.1.6 #define NV_ENC_EVENT_PARAMS_VER NVENCAPI_STRUCT_VERSION(1)

Macro for constructing the version field of _NV_ENC_EVENT_PARAMS

4.1.1.7 #define NV_ENC_INITIALIZE_PARAMS_VER (NVENCAPI_STRUCT_VERSION(5) | (1<<31))

macro for constructing the version field of _NV_ENC_INITIALIZE_PARAMS

4.1.1.8 #define NV_ENC_LOCK_BITSTREAM_VER NVENCAPI_STRUCT_VERSION(1)

Macro for constructing the version field of _NV_ENC_LOCK_BITSTREAM

4.1.1.9 #define NV_ENC_LOCK_INPUT_BUFFER_VER NVENCAPI_STRUCT_VERSION(1)

Macro for constructing the version field of _NV_ENC_LOCK_INPUT_BUFFER

4.1.1.10 #define NV_ENC_MAP_INPUT_RESOURCE_VER NVENCAPI_STRUCT_VERSION(4)

Macro for constructing the version field of _NV_ENC_MAP_INPUT_RESOURCE

4.1.1.11 #define NV_ENC_MEONLY_PARAMS_VER NVENCAPI_STRUCT_VERSION(2)

NV_ENC_MEONLY_PARAMS struct version

4.1.1.12 #define NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS_VER NVENCAPI_STRUCT_-VERSION(1)

Macro for constructing the version field of _NV_ENC_OPEN_ENCODE_SESSIONEX_PARAMS

4.1.1.13 #define NV_ENC_PARAMS_RC_CBR2 NV_ENC_PARAMS_RC_CBR

Deprecated

4.1.1.14 #define NV_ENC_PIC_PARAMS_VER (NVENCAPI_STRUCT_VERSION(4) | (1<<31))

Macro for constructing the version field of _NV_ENC_PIC_PARAMS

4.1.1.15 #define NV_ENC_PRESET_CONFIG_VER (NVENCAPI_STRUCT_VERSION(4) | (1<<31))

macro for constructing the version field of _NV_ENC_PRESET_CONFIG

4.1.1.16 #define NV_ENC_RC_PARAMS_VER NVENCAPI_STRUCT_VERSION(1)

macro for constructing the version field of _NV_ENC_RC_PARAMS

4.1.1.17 #define NV_ENC_RECONFIGURE_PARAMS_VER (NVENCAPI_STRUCT_VERSION(1) | (1<<31))

macro for constructing the version field of _NV_ENC_RECONFIGURE_PARAMS

4.1.1.18 #define NV_ENC_REGISTER_RESOURCE_VER NVENCAPI_STRUCT_VERSION(3)

Macro for constructing the version field of _NV_ENC_REGISTER_RESOURCE

4.1.1.19 #define NV_ENC_SEQUENCE_PARAM_PAYLOAD_VER NVENCAPI_STRUCT_VERSION(1)

Macro for constructing the version field of _NV_ENC_SEQUENCE_PARAM_PAYLOAD

4.1.1.20 #define NV_ENC_STAT_VER NVENCAPI_STRUCT_VERSION(1)

Macro for constructing the version field of _NV_ENC_STAT

4.1.2 Enumeration Type Documentation

4.1.2.1 enum NV_ENC_BUFFER_FORMAT

Input buffer formats

Enumerator:

NV_ENC_BUFFER_FORMAT_UNDEFINED Undefined buffer format

NV_ENC_BUFFER_FORMAT_NV12 Semi-Planar YUV [Y plane followed by interleaved UV plane]

NV_ENC_BUFFER_FORMAT_YV12 Planar YUV [Y plane followed by V and U planes]

NV_ENC_BUFFER_FORMAT_IYUV Planar YUV [Y plane followed by U and V planes]

NV_ENC_BUFFER_FORMAT_YUV444 Planar YUV [Y plane followed by U and V planes]

NV_ENC_BUFFER_FORMAT_YUV420_10BIT 10 bit Semi-Planar YUV [Y plane followed by interleaved UV plane]. Each pixel of size 2 bytes. Most Significant 10 bits contain pixel data.

NV_ENC_BUFFER_FORMAT_YUV444_10BIT 10 bit Planar YUV444 [Y plane followed by U and V planes]. Each pixel of size 2 bytes. Most Significant 10 bits contain pixel data.

NV_ENC_BUFFER_FORMAT_ARGB 8 bit Packed A8R8G8B8

NV_ENC_BUFFER_FORMAT_ARGB10 10 bit Packed A2R10G10B10. Each pixel of size 2 bytes. Most Significant 10 bits contain pixel data.

NV_ENC_BUFFER_FORMAT_AYUV 8 bit Packed A8Y8U8V8

NV_ENC_BUFFER_FORMAT_ABGR 8 bit Packed A8B8G8R8

NV_ENC_BUFFER_FORMAT_ABGR10 10 bit Packed A2B10G10R10. Each pixel of size 2 bytes. Most Significant 10 bits contain pixel data.

4.1.2.2 enum NV_ENC_CAPS

Encoder capabilities enumeration.

Enumerator:

NV_ENC_CAPS_NUM_MAX_BFRAMES Maximum number of B-Frames supported.

NV_ENC_CAPS_SUPPORTED_RATECONTROL_MODES Rate control modes supported.
The API return value is a bitmask of the values in NV_ENC_PARAMS_RC_MODE.

NV_ENC_CAPS_SUPPORT_FIELD_ENCODING Indicates HW support for field mode encoding.
0 : Interlaced mode encoding is not supported.
1 : Interlaced field mode encoding is supported.
2 : Interlaced frame encoding and field mode encoding are both supported.

NV_ENC_CAPS_SUPPORT_MONOCHROME Indicates HW support for monochrome mode encoding.
0 : Monochrome mode not supported.
1 : Monochrome mode supported.

NV_ENC_CAPS_SUPPORT_FMO Indicates HW support for FMO.
0 : FMO not supported.
1 : FMO supported.

NV_ENC_CAPS_SUPPORT_QPELMV Indicates HW capability for Quarter pel motion estimation.
0 : QuarterPel Motion Estimation not supported.
1 : QuarterPel Motion Estimation supported.

NV_ENC_CAPS_SUPPORT_BDIRECT_MODE H.264 specific. Indicates HW support for BDirect modes.
0 : BDirect mode encoding not supported.
1 : BDirect mode encoding supported.

NV_ENC_CAPS_SUPPORT_CABAC H264 specific. Indicates HW support for CABAC entropy coding mode.
0 : CABAC entropy coding not supported.
1 : CABAC entropy coding supported.

NV_ENC_CAPS_SUPPORT_ADAPTIVE_TRANSFORM Indicates HW support for Adaptive Transform.
0 : Adaptive Transform not supported.
1 : Adaptive Transform supported.

NV_ENC_CAPS_SUPPORT_RESERVED Reserved enum field.

NV_ENC_CAPS_NUM_MAX_TEMPORAL_LAYERS Indicates HW support for encoding Temporal layers.
0 : Encoding Temporal layers not supported.
1 : Encoding Temporal layers supported.

NV_ENC_CAPS_SUPPORT_HIERARCHICAL_PFRAMES Indicates HW support for Hierarchical P frames.
0 : Hierarchical P frames not supported.
1 : Hierarchical P frames supported.

NV_ENC_CAPS_SUPPORT_HIERARCHICAL_BFRAMES Indicates HW support for Hierarchical B frames.
0 : Hierarchical B frames not supported.
1 : Hierarchical B frames supported.

NV_ENC_CAPS_LEVEL_MAX Maximum Encoding level supported (See NV_ENC_LEVEL for details).

NV_ENC_CAPS_LEVEL_MIN Minimum Encoding level supported (See NV_ENC_LEVEL for details).

NV_ENC_CAPS_SEPARATE_COLOUR_PLANE Indicates HW support for separate colour plane encoding.

0 : Separate colour plane encoding not supported.

1 : Separate colour plane encoding supported.

NV_ENC_CAPS_WIDTH_MAX Maximum output width supported.

NV_ENC_CAPS_HEIGHT_MAX Maximum output height supported.

NV_ENC_CAPS_SUPPORT_TEMPORAL_SVC Indicates Temporal Scalability Support.

0 : Temporal SVC encoding not supported.

1 : Temporal SVC encoding supported.

NV_ENC_CAPS_SUPPORT_DYN_RES_CHANGE Indicates Dynamic Encode Resolution Change Support. Support added from NvEncodeAPI version 2.0.

0 : Dynamic Encode Resolution Change not supported.

1 : Dynamic Encode Resolution Change supported.

NV_ENC_CAPS_SUPPORT_DYN_BITRATE_CHANGE Indicates Dynamic Encode Bitrate Change Support. Support added from NvEncodeAPI version 2.0.

0 : Dynamic Encode bitrate change not supported.

1 : Dynamic Encode bitrate change supported.

NV_ENC_CAPS_SUPPORT_DYN_FORCE_CONSTQP Indicates Forcing Constant QP On The Fly Support. Support added from NvEncodeAPI version 2.0.

0 : Forcing constant QP on the fly not supported.

1 : Forcing constant QP on the fly supported.

NV_ENC_CAPS_SUPPORT_DYN_RCMODE_CHANGE Indicates Dynamic rate control mode Change Support.

0 : Dynamic rate control mode change not supported.

1 : Dynamic rate control mode change supported.

NV_ENC_CAPS_SUPPORT_SUBFRAME_READBACK Indicates Subframe readback support for slice-based encoding.

0 : Subframe readback not supported.

1 : Subframe readback supported.

NV_ENC_CAPS_SUPPORT_CONSTRAINED_ENCODING Indicates Constrained Encoding mode support. Support added from NvEncodeAPI version 2.0.

0 : Constrained encoding mode not supported.

1 : Constarined 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 bitmask of the values defined in NV_ENC_PREPROC_FLAGS

NV_ENC_CAPS_ASYNC_ENCODE_SUPPORT Indicates support Async mode.

0 : Async Encode mode not supported.

1 : Async Encode mode supported.

NV_ENC_CAPS_MB_NUM_MAX Maximum MBs per frame supported.

NV_ENC_CAPS_MB_PER_SEC_MAX Maximum aggregate throughput in MBs per sec.

NV_ENC_CAPS_SUPPORT_YUV444_ENCODE Indicates HW support for YUV444 mode encoding.

0 : YUV444 mode encoding not supported.

1 : YUV444 mode encoding supported.

NV_ENC_CAPS_SUPPORT_LOSSLESS_ENCODE Indicates HW support for lossless encoding.

0 : lossless encoding not supported.

1 : lossless encoding supported.

NV_ENC_CAPS_SUPPORT_SAO Indicates HW support for Sample Adaptive Offset.

0 : SAO not supported.

1 : SAO encoding supported.

NV_ENC_CAPS_SUPPORT_MEONLY_MODE Indicates HW support for MEOnly Mode.

0 : MEOnly Mode not supported.

1 : MEOnly Mode supported.

NV_ENC_CAPS_SUPPORT_LOOKAHEAD Indicates HW support for lookahead encoding (enableLookahead=1).

0 : Lookahead not supported.

1 : Lookahead supported.

NV_ENC_CAPS_SUPPORT_TEMPORAL_AQ Indicates HW support for temporal AQ encoding (enableTemporalAQ=1).

0 : Temporal AQ not supported.

1 : Temporal AQ supported.

NV_ENC_CAPS_SUPPORT_10BIT_ENCODE Indicates HW support for 10 bit encoding.

0 : 10 bit encoding not supported.

1 : 10 bit 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_CUSIZE

HEVC CU SIZE

4.1.2.9 enum NV_ENC_INPUT_RESOURCE_TYPE

Input Resource type

Enumerator:

NV_ENC_INPUT_RESOURCE_TYPE_DIRECTX input resource type is a directx9 surface

NV_ENC_INPUT_RESOURCE_TYPE_CUDADEVICEPTR input resource type is a cuda device pointer surface

NV_ENC_INPUT_RESOURCE_TYPE_CUDAARRAY input resource type is a cuda array surface

4.1.2.10 enum NV_ENC_LEVEL

Encoding levels

4.1.2.11 enum NV_ENC_MEMORY_HEAP

Memory heap to allocate input and output buffers.

Enumerator:

NV_ENC_MEMORY_HEAP_AUTOSELECT Memory heap to be decided by the encoder driver based on the usage

NV_ENC_MEMORY_HEAP_VID Memory heap is in local video memory

NV_ENC_MEMORY_HEAP_SYSMEM_CACHED Memory heap is in cached system memory

NV_ENC_MEMORY_HEAP_SYSMEM_UNCACHED Memory heap is in uncached system memory

4.1.2.12 enum NV_ENC_MV_PRECISION

Motion vector precisions

Enumerator:

NV_ENC_MV_PRECISION_DEFAULT Driver selects QuarterPel motion vector precision by default

NV_ENC_MV_PRECISION_FULL_PEL FullPel motion vector precision

NV_ENC_MV_PRECISION_HALF_PEL HalfPel motion vector precision

NV_ENC_MV_PRECISION_QUARTER_PEL QuarterPel motion vector precision

4.1.2.13 enum NV_ENC_PARAMS_FRAME_FIELD_MODE

Input frame encode modes

Enumerator:

NV_ENC_PARAMS_FRAME_FIELD_MODE_FRAME Frame mode

NV_ENC_PARAMS_FRAME_FIELD_MODE_FIELD Field mode

NV_ENC_PARAMS_FRAME_FIELD_MODE_MBAFF MB adaptive frame/field

4.1.2.14 enum NV_ENC_PARAMS_RC_MODE

Rate Control Modes

Enumerator:

- ***NV_ENC_PARAMS_RC_CONSTQP*** Constant QP mode
- ***NV_ENC_PARAMS_RC_VBR*** Variable bitrate mode
- ***NV_ENC_PARAMS_RC_CBR*** Constant bitrate mode
- ***NV_ENC_PARAMS_RC_VBR_MINQP*** Variable bitrate mode with MinQP
- ***NV_ENC_PARAMS_RC_2_PASS_QUALITY*** Multi pass encoding optimized for image quality and works only with low latency mode
- ***NV_ENC_PARAMS_RC_2_PASS_FRAMESIZE_CAP*** Multi pass encoding optimized for maintaining frame size and works only with low latency mode
- ***NV_ENC_PARAMS_RC_2_PASS_VBR*** Multi pass VBR

4.1.2.15 enum NV_ENC_PIC_FLAGS

Encode Picture encode flags.

Enumerator:

- ***NV_ENC_PIC_FLAG_FORCEINTRA*** Encode the current picture as an Intra picture
- ***NV_ENC_PIC_FLAG_FORCEIDR*** Encode the current picture as an IDR picture. This flag is only valid when Picture type decision is taken by the Encoder [_NV_ENC_INITIALIZE_PARAMS::enablePTD == 1].
- ***NV_ENC_PIC_FLAG_OUTPUT_SPSPPS*** Write the sequence and picture header in encoded bitstream of the current picture
- ***NV_ENC_PIC_FLAG_EOS*** Indicates end of the input stream

4.1.2.16 enum NV_ENC_PIC_STRUCT

Input picture structure

Enumerator:

- ***NV_ENC_PIC_STRUCT_FRAME*** Progressive frame
- ***NV_ENC_PIC_STRUCT_FIELD_TOP_BOTTOM*** Field encoding top field first
- ***NV_ENC_PIC_STRUCT_FIELD_BOTTOM_TOP*** Field encoding bottom field first

4.1.2.17 enum NV_ENC_PIC_TYPE

Input picture type

Enumerator:

- ***NV_ENC_PIC_TYPE_P*** Forward predicted
- ***NV_ENC_PIC_TYPE_B*** Bi-directionally predicted picture
- ***NV_ENC_PIC_TYPE_I*** Intra predicted picture

NV_ENC_PIC_TYPE_IDR IDR picture

NV_ENC_PIC_TYPE_BI Bi-directionally predicted with only Intra MBs

NV_ENC_PIC_TYPE_SKIPPED Picture is skipped

NV_ENC_PIC_TYPE_INTRA_REFRESH First picture in intra refresh cycle

NV_ENC_PIC_TYPE_UNKNOWN Picture type unknown

4.1.2.18 enum NV_ENC_STEREO_PACKING_MODE

Stereo frame packing modes.

Enumerator:

NV_ENC_STEREO_PACKING_MODE_NONE No Stereo packing required

NV_ENC_STEREO_PACKING_MODE_CHECKERBOARD Checkerboard mode for packing stereo frames

NV_ENC_STEREO_PACKING_MODE_COLINTERLEAVE Column Interleave mode for packing stereo frames

NV_ENC_STEREO_PACKING_MODE_ROWINTERLEAVE Row Interleave mode for packing stereo frames

NV_ENC_STEREO_PACKING_MODE_SIDEBYSIDE Side-by-side mode for packing stereo frames

NV_ENC_STEREO_PACKING_MODE_TOPBOTTOM Top-Bottom mode for packing stereo frames

NV_ENC_STEREO_PACKING_MODE_FRAMESEQ Frame Sequential mode for packing stereo frames

4.1.2.19 enum NVENCSTATUS

Error Codes

Enumerator:

NV_ENC_SUCCESS This indicates that API call returned with no errors.

NV_ENC_ERR_NO_ENCODE_DEVICE This indicates that no encode capable devices were detected.

NV_ENC_ERR_UNSUPPORTED_DEVICE This indicates that devices pass by the client is not supported.

NV_ENC_ERR_INVALID_ENCODERDEVICE This indicates that the encoder device supplied by the client is not valid.

NV_ENC_ERR_INVALID_DEVICE This indicates that device passed to the API call is invalid.

NV_ENC_ERR_DEVICE_NOT_EXIST This indicates that device passed to the API call is no longer available and needs to be reinitialized. The clients need to destroy the current encoder session by freeing the allocated input output buffers and destroying the device and create a new encoding session.

NV_ENC_ERR_INVALID_PTR This indicates that one or more of the pointers passed to the API call is invalid.

NV_ENC_ERR_INVALID_EVENT This indicates that completion event passed in NvEncEncodePicture() call is invalid.

NV_ENC_ERR_INVALID_PARAM This indicates that one or more of the parameter passed to the API call is invalid.

NV_ENC_ERR_INVALID_CALL This indicates that an API call was made in wrong sequence/order.

NV_ENC_ERR_OUT_OF_MEMORY This indicates that the API call failed because it was unable to allocate enough memory to perform the requested operation.

NV_ENC_ERR_ENCODER_NOT_INITIALIZED This indicates that the encoder has not been initialized with NvEncInitializeEncoder() or that initialization has failed. The client cannot allocate input or output buffers or do any encoding related operation before successfully initializing the encoder.

NV_ENC_ERR_UNSUPPORTED_PARAM This indicates that an unsupported parameter was passed by the client.

NV_ENC_ERR_LOCK_BUSY This indicates that the NvEncLockBitstream() failed to lock the output buffer. This happens when the client makes a non blocking lock call to access the output bitstream by passing NV_ENC_LOCK_BITSTREAM::doNotWait flag. This is not a fatal error and client should retry the same operation after few milliseconds.

NV_ENC_ERR_NOT_ENOUGH_BUFFER This indicates that the size of the user buffer passed by the client is insufficient for the requested operation.

NV_ENC_ERR_INVALID_VERSION This indicates that an invalid struct version was used by the client.

NV_ENC_ERR_MAP_FAILED This indicates that NvEncMapInputResource() API failed to map the client provided input resource.

NV_ENC_ERR_NEED_MORE_INPUT This indicates encode driver requires more input buffers to produce an output bitstream. If this error is returned from NvEncEncodePicture() API, this is not a fatal error. If the client is encoding with B frames then, NvEncEncodePicture() API might be buffering the input frame for re-ordering.

A client operating in synchronous mode cannot call NvEncLockBitstream() API on the output bitstream buffer if NvEncEncodePicture() returned the NV_ENC_ERR_NEED_MORE_INPUT error code. The client must continue providing input frames until encode driver returns NV_ENC_SUCCESS. After receiving NV_ENC_SUCCESS status the client can call NvEncLockBitstream() API on the output buffers in the same order in which it has called NvEncEncodePicture().

NV_ENC_ERR_ENCODER_BUSY This indicates that the HW encoder is busy encoding and is unable to encode the input. The client should call NvEncEncodePicture() again after few milliseconds.

NV_ENC_ERR_EVENT_NOT_REGISTERD This indicates that the completion event passed in NvEncEncodePicture() API has not been registered with encoder driver using NvEncRegisterAsyncEvent().

NV_ENC_ERR_GENERIC This indicates that an unknown internal error has occurred.

NV_ENC_ERR_INCOMPATIBLE_CLIENT_KEY This indicates that the client is attempting to use a feature that is not available for the license type for the current system.

NV_ENC_ERR_UNIMPLEMENTED This indicates that the client is attempting to use a feature that is not implemented for the current version.

NV_ENC_ERR_RESOURCE_REGISTER_FAILED This indicates that the NvEncRegisterResource API failed to register the resource.

NV_ENC_ERR_RESOURCE_NOT_REGISTERED This indicates that the client is attempting to unregister a resource that has not been successfully registered.

NV_ENC_ERR_RESOURCE_NOT_MAPPED This indicates that the client is attempting to unmap a resource that has not been successfully mapped.

4.2 NvEncodeAPI Functions

Functions

- NVENCSTATUS NVENCAPI NvEncOpenEncodeSession (void ∗device, uint32_t deviceType, void ∗∗encoder)

  Opens an encoding session.

- NVENCSTATUS NVENCAPI NvEncGetEncodeGUIDCount (void ∗encoder, uint32_t ∗encodeGUIDCount)

  Retrieves the number of supported encode GUIDs.

- NVENCSTATUS NVENCAPI NvEncGetEncodeGUIDs (void ∗encoder, GUID ∗GUIDs, uint32_t guidArraySize, uint32_t ∗GUIDCount)

  Retrieves an array of supported encoder codec GUIDs.

- NVENCSTATUS NVENCAPI NvEncGetEncodeProfileGUIDCount (void ∗encoder, GUID encodeGUID, uint32_t ∗encodeProfileGUIDCount)

  Retrieves the number of supported profile GUIDs.

- NVENCSTATUS NVENCAPI NvEncGetEncodeProfileGUIDs (void ∗encoder, GUID encodeGUID, GUID ∗profileGUIDs, uint32_t guidArraySize, uint32_t ∗GUIDCount)

  Retrieves an array of supported encode profile GUIDs.

- NVENCSTATUS NVENCAPI NvEncGetInputFormatCount (void ∗encoder, GUID encodeGUID, uint32_t ∗inputFmtCount)

  Retrieve the number of supported Input formats.

- NVENCSTATUS NVENCAPI NvEncGetInputFormats (void ∗encoder, GUID encodeGUID, NV_ENC_BUFFER_FORMAT ∗inputFmts, uint32_t inputFmtArraySize, uint32_t ∗inputFmtCount)

  Retrieves an array of supported Input formats.

- NVENCSTATUS NVENCAPI NvEncGetEncodeCaps (void ∗encoder, GUID encodeGUID, NV_ENC_CAPS_PARAM ∗capsParam, int ∗capsVal)

  Retrieves the capability value for a specified encoder attribute.

- NVENCSTATUS NVENCAPI NvEncGetEncodePresetCount (void ∗encoder, GUID encodeGUID, uint32_t ∗encodePresetGUIDCount)

  Retrieves the number of supported preset GUIDs.

- NVENCSTATUS NVENCAPI NvEncGetEncodePresetGUIDs (void ∗encoder, GUID encodeGUID, GUID ∗presetGUIDs, uint32_t guidArraySize, uint32_t ∗encodePresetGUIDCount)

  Receives an array of supported encoder preset GUIDs.

- NVENCSTATUS NVENCAPI NvEncGetEncodePresetConfig (void ∗encoder, GUID encodeGUID, GUID presetGUID, NV_ENC_PRESET_CONFIG ∗presetConfig)

  Returns a preset config structure supported for given preset GUID.

- NVENCSTATUS NVENCAPI NvEncInitializeEncoder (void ∗encoder, NV_ENC_INITIALIZE_PARAMS ∗createEncodeParams)

  Initialize the encoder.

- NVENCSTATUS NVENCAPI NvEncCreateInputBuffer (void ∗encoder, NV_ENC_CREATE_INPUT_-BUFFER ∗createInputBufferParams)

  Allocates Input buffer.

- NVENCSTATUS NVENCAPI NvEncDestroyInputBuffer (void ∗encoder, NV_ENC_INPUT_PTR input-Buffer)

  Release an input buffers.

- NVENCSTATUS NVENCAPI NvEncCreateBitstreamBuffer (void ∗encoder, NV_ENC_CREATE_-BITSTREAM_BUFFER ∗createBitstreamBufferParams)

  Allocates an output bitstream buffer.

- NVENCSTATUS NVENCAPI NvEncDestroyBitstreamBuffer (void ∗encoder, NV_ENC_OUTPUT_PTR bit-streamBuffer)

  Release a bitstream buffer.

- NVENCSTATUS NVENCAPI NvEncEncodePicture (void ∗encoder, NV_ENC_PIC_PARAMS ∗encodePicParams)

  Submit an input picture for encoding.

- NVENCSTATUS NVENCAPI NvEncLockBitstream (void ∗encoder, NV_ENC_LOCK_BITSTREAM ∗lockBitstreamBufferParams)

  Lock output bitstream buffer.

- NVENCSTATUS NVENCAPI NvEncUnlockBitstream (void ∗encoder, NV_ENC_OUTPUT_PTR bitstream-Buffer)

  Unlock the output bitstream buffer.

- NVENCSTATUS NVENCAPI NvEncLockInputBuffer (void ∗encoder, NV_ENC_LOCK_INPUT_BUFFER ∗lockInputBufferParams)

  Locks an input buffer.

- NVENCSTATUS NVENCAPI NvEncUnlockInputBuffer (void ∗encoder, NV_ENC_INPUT_PTR input-Buffer)

  Unlocks the input buffer.

- NVENCSTATUS NVENCAPI NvEncGetEncodeStats (void ∗encoder, NV_ENC_STAT ∗encodeStats)

  Get encoding statistics.

- NVENCSTATUS NVENCAPI NvEncGetSequenceParams (void ∗encoder, NV_ENC_SEQUENCE_PARAM_-PAYLOAD ∗sequenceParamPayload)

  Get encoded sequence and picture header.

- NVENCSTATUS NVENCAPI NvEncRegisterAsyncEvent (void ∗encoder, NV_ENC_EVENT_PARAMS ∗eventParams)

  Register event for notification to encoding completion.

- NVENCSTATUS NVENCAPI NvEncUnregisterAsyncEvent (void ∗encoder, NV_ENC_EVENT_PARAMS ∗eventParams)

  Unregister completion event.

- NVENCSTATUS NVENCAPI NvEncMapInputResource (void ∗encoder, NV_ENC_MAP_INPUT_-RESOURCE ∗mapInputResParams)

  Map an externally created input resource pointer for encoding.

- NVENCSTATUS NVENCAPI NvEncUnmapInputResource (void ∗encoder, NV_ENC_INPUT_PTR mapped-InputBuffer)

  UnMaps a NV_ENC_INPUT_PTR which was mapped for encoding.

- NVENCSTATUS NVENCAPI NvEncDestroyEncoder (void ∗encoder)

  Destroy Encoding Session.

- NVENCSTATUS NVENCAPI NvEncInvalidateRefFrames (void ∗encoder, uint64_t invalidRefFrameTimeS-tamp)

  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 reg-isteredResource)

  Unregisters a resource previously registered with the Nvidia Video Encoder Interface.

- NVENCSTATUS NVENCAPI NvEncReconfigureEncoder (void ∗encoder, NV_ENC_RECONFIGURE_-PARAMS ∗reInitEncodeParams)

  Reconfigure an existing encoding session.

- NVENCSTATUS NVENCAPI NvEncCreateMVBuffer (void ∗encoder, NV_ENC_CREATE_MV_BUFFER ∗createMVBufferParams)

  Allocates output MV buffer for ME only mode.

- NVENCSTATUS NVENCAPI NvEncDestroyMVBuffer (void ∗encoder, NV_ENC_OUTPUT_PTR mvBuffer)

  Release an output MV buffer for ME only mode.

- NVENCSTATUS NVENCAPI NvEncRunMotionEstimationOnly (void ∗encoder, NV_ENC_MEONLY_-PARAMS ∗meOnlyParams)

  Submit an input picture and reference frame for motion estimation in ME only mode.

- NVENCSTATUS NVENCAPI NvEncodeAPIGetMaxSupportedVersion (uint32_t ∗version)

  Get the largest NvEncodeAPI version supported by the driver.

- 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 bitsteam data by locking the `bitstreamBuffer` using the NvEncLockBitstream() function.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***createBitstreamBufferParams*** Pointer NV_ENC_CREATE_BITSTREAM_BUFFER for details.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.2 NVENCSTATUS NVENCAPI NvEncCreateInputBuffer (void $*$ *encoder*, NV_ENC_CREATE_INPUT_BUFFER $*$ *createInputBufferParams*)

This function is used to allocate an input buffer. The client must enumerate the input buffer format before allocating the input buffer resources. The NV_ENC_INPUT_PTR returned by the NvEncodeAPI interface in the NV_ENC_-CREATE_INPUT_BUFFER::inputBuffer field can be directly used in NvEncEncodePicture() API. The number of input buffers to be allocated by the client must be at least 4 more than the number of B frames being used for encoding.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***createInputBufferParams*** Pointer to the NV_ENC_CREATE_INPUT_BUFFER structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_GENERIC

4.2.1.3 NVENCSTATUS NVENCAPI NvEncCreateMVBuffer (void ∗ *encoder*, NV_ENC_CREATE_MV_BUFFER ∗ *createMVBufferParams*)

This function is used to allocate an output MV buffer. The size of the mvBuffer is dependent on the frame height and width of the last NvEncCreateInputBuffer() call. The NV_ENC_OUTPUT_PTR returned by the NvEncodeAPI interface in the NV_ENC_CREATE_MV_BUFFER::mvBuffer field should be used in NvEncRunMotionEstimationOnly() API. Client must lock NV_ENC_CREATE_MV_BUFFER::mvBuffer using NvEncLockBitstream() API to get the motion vector data.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***createMVBufferParams*** Pointer to the NV_ENC_CREATE_MV_BUFFER structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_GENERIC

4.2.1.4 NVENCSTATUS NVENCAPI NvEncDestroyBitstreamBuffer (void ∗ *encoder*, NV_ENC_OUTPUT_PTR *bitstreamBuffer*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***bitstreamBuffer*** Pointer to the bitstream buffer being released.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.5 NVENCSTATUS NVENCAPI NvEncDestroyEncoder (void ∗ *encoder*)

Destroys the encoder session previously created using NvEncOpenEncodeSession() function. The client must flush the encoder before freeing any resources. In order to flush the encoder the client must pass a NULL encode picture packet and either wait for the NvEncEncodePicture() function to return in synchronous mode or wait for the flush event to be signaled by the encoder in asynchronous mode. The client must free all the input and output resources created using the NvEncodeAPI interface before destroying the encoder. If the client is operating in asynchronous mode, it must also unregister the completion events previously registered.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.6 NVENCSTATUS NVENCAPI NvEncDestroyInputBuffer (void ∗ *encoder*, NV_ENC_INPUT_PTR *inputBuffer*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***inputBuffer*** Pointer to the input buffer to be released.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_GENERIC

4.2.1.7 NVENCSTATUS NVENCAPI NvEncDestroyMVBuffer (void ∗ *encoder*, NV_ENC_OUTPUT_PTR *mvBuffer*)

This function is used to release the output MV buffer allocated using the NvEncCreateMVBuffer() function. The client must release the output mvBuffer using this function before destroying the encoder session.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***mvBuffer*** Pointer to the mvBuffer being released.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.8 NVENCSTATUS NVENCAPI NvEncEncodePicture (void ∗ *encoder*, NV_ENC_PIC_PARAMS ∗ *encodePicParams*)

This function is used to submit an input picture buffer for encoding. The encoding parameters are passed using `*encodePicParams` which is a pointer to the _NV_ENC_PIC_PARAMS structure.

If the client has set NV_ENC_INITIALIZE_PARAMS::enablePTD to 0, then it must send a valid value for the following fields.

- NV_ENC_PIC_PARAMS::pictureType
- NV_ENC_PIC_PARAMS_H264::displayPOCSyntax (H264 only)
- NV_ENC_PIC_PARAMS_H264::frameNumSyntax(H264 only)
- NV_ENC_PIC_PARAMS_H264::refPicFlag(H264 only)

Asynchronous Encoding

If the client has enabled asynchronous mode of encoding by setting NV_ENC_INITIALIZE_PARAMS::enableEncodeAsync to 1 in the NvEncInitializeEncoder() API ,then the client must send a valid NV_ENC_PIC_PARAMS::completionEvent. Incase 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
 buffersfor 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_SUCESS. Both
the submission are part of the same ::NvEncEncodePicture() function call.
The client can now read the encoded data for both the frames by locking the output
O2 followed by  O3 ,by calling ::NvEncLockBitstream() API.

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***encodePicParams*** Pointer to the _NV_ENC_PIC_PARAMS structure.

Returns:

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

4.2.1.9 NVENCSTATUS NVENCAPI NvEncGetEncodeCaps (void * *encoder*, GUID *encodeGUID*, NV_ENC_CAPS_PARAM * *capsParam*, int * *capsVal*)

The function returns the capability value for a given encoder attribute. The client must validate the encodeGUID using NvEncGetEncodeGUIDs() API before calling this function. The encoder attribute being queried are enumerated in NV_ENC_CAPS_PARAM enum.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***encodeGUID*** Encode GUID, corresponding to which the capability attribute is to be retrieved.

← ***capsParam*** Used to specify attribute being queried. Refer NV_ENC_CAPS_PARAM for more details.

→ ***capsVal*** The value corresponding to the capability attribute being queried.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

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

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

→ ***encodeGUIDCount*** Number of supported encode GUIDs.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

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

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***guidArraySize*** Number of GUIDs to retrieved. Should be set to the number retrieved using NvEncGetEncodeGUIDCount.

→ ***GUIDs*** Array of supported Encode GUIDs.

→ ***GUIDCount*** Number of supported Encode GUIDs.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.12 NVENCSTATUS NVENCAPI NvEncGetEncodePresetConfig (void ∗ *encoder*, GUID *encodeGUID*, GUID *presetGUID*, NV_ENC_PRESET_CONFIG ∗ *presetConfig*)

The function returns a preset config structure for a given preset guid. Before using this function the client must enumerate the preset 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

4.2.1.13 NVENCSTATUS NVENCAPI NvEncGetEncodePresetCount (void ∗ *encoder*, GUID *encodeGUID*, uint32_t ∗ *encodePresetGUIDCount*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***encodeGUID*** Encode GUID, corresponding to which the number of supported presets is to be retrieved.

→ ***encodePresetGUIDCount*** Receives the number of supported preset GUIDs.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.14 NVENCSTATUS NVENCAPI NvEncGetEncodePresetGUIDs (void ∗ *encoder*, GUID *encodeGUID*, GUID ∗ *presetGUIDs*, uint32_t *guidArraySize*, uint32_t ∗ *encodePresetGUIDCount*)

The function returns an array of encode preset guids available for a given codec. The client can directly use one of the preset guids based upon the use case or target device. The preset guid chosen can be directly used in NV_-ENC_INITIALIZE_PARAMS::presetGUID parameter to NvEncEncodePicture() API. Alternately client can also use the preset guid 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 guids passed in `preset` GUIDs

→ ***presetGUIDs*** Array of supported Encode preset GUIDs from the NvEncodeAPI interface to client.

→ ***encodePresetGUIDCount*** Receives the number of preset GUIDs returned by the NvEncodeAPI interface.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.15 NVENCSTATUS 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 guids supported by the NvEncodeAPI interface. After determining the codec guid, it can query the NvEncodeAPI interface to determine the number of profile guids supported for a particular codec guid.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***encodeGUID*** The codec guid for which the profile guids are being enumerated.

→ ***encodeProfileGUIDCount*** Number of encode profiles supported for the given encodeGUID.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

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

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***encodeGUID*** The encode guid whose profile guids are being enumerated.

← ***guidArraySize*** Number of GUIDs to be retrieved. Should be set to the number retrieved using NvEncGetEncodeProfileGUIDCount.

→ ***profileGUIDs*** Array of supported Encode Profile GUIDs

→ ***GUIDCount*** Number of valid encode profile GUIDs in `profileGUIDs` array.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.17 NVENCSTATUS NVENCAPI NvEncGetEncodeStats (void ∗ *encoder*, NV_ENC_STAT ∗ *encodeStats*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

$\leftrightarrow$ ***encodeStats*** Pointer to the _NV_ENC_STAT structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.18 NVENCSTATUS NVENCAPI NvEncGetInputFormatCount (void $*$ *encoder*, GUID *encodeGUID*, uint32_t $*$ *inputFmtCount*)

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

Parameters:

$\leftarrow$ ***encoder*** Pointer to the NvEncodeAPI interface.

$\leftarrow$ ***encodeGUID*** Encode GUID, corresponding to which the number of supported input formats is to be retrieved.

$\rightarrow$ ***inputFmtCount*** Number of input formats supported for specified Encode GUID.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.19 NVENCSTATUS NVENCAPI NvEncGetInputFormats (void $*$ *encoder*, GUID *encodeGUID*, NV_ENC_BUFFER_FORMAT $*$ *inputFmts*, uint32_t *inputFmtArraySize*, uint32_t $*$ *inputFmtCount*)

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

Parameters:

$\leftarrow$ ***encoder*** Pointer to the NvEncodeAPI interface.

$\leftarrow$ ***encodeGUID*** Encode GUID, corresponding to which the number of supported input formats is to be retrieved.

$\leftarrow$ ***inputFmtArraySize*** Size input format count array passed in `inputFmts`.

$\rightarrow$ ***inputFmts*** Array of input formats supported for this Encode GUID.

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

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.20 NVENCSTATUS NVENCAPI NvEncGetSequenceParams (void ∗ *encoder*, NV_ENC_SEQUENCE_PARAM_PAYLOAD ∗ *sequenceParamPayload*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***sequenceParamPayload*** Pointer to the _NV_ENC_SEQUENCE_PARAM_PAYLOAD structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.21 NVENCSTATUS NVENCAPI NvEncInitializeEncoder (void ∗ *encoder*, NV_ENC_INITIALIZE_PARAMS ∗ *createEncodeParams*)

This API must be used to initialize the encoder. The initialization parameter is passed using `*createEncodeParams` The client must send the following fields of the _NV_ENC_INITIALIZE_PARAMS structure with a valid value.

- NV_ENC_INITIALIZE_PARAMS::encodeGUID

- NV_ENC_INITIALIZE_PARAMS::encodeWidth
- NV_ENC_INITIALIZE_PARAMS::encodeHeight

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

There are two modes of operation for the encoder namely:

- Asynchronous mode
- Synchronous mode

The client can select asynchronous or synchronous mode by setting the `enableEncodeAsync` field in _NV_ENC_-INITIALIZE_PARAMS to 1 or 0 respectively.

Asynchronous mode of operation:

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

NOTE: Asynchronous mode is not supported on Linux.

Synchronous mode of operation:

The client can select synchronous mode by setting NV_ENC_INITIALIZE_PARAMS::enableEncodeAsync to 0. The client working in synchronous mode can work in a single threaded or multi threaded mode. The client need not allocate any event objects. The client can only lock the bitstream data after NvEncodeAPI interface has returned NV_ENC_SUCCESS from encode picture. The NvEncodeAPI interface can return NV_ENC_ERR_-NEED_MORE_INPUT error code from NvEncEncodePicture() API. The client must not lock the output buffer in such case but should send the next frame for encoding. The client must keep on calling NvEncEncodePicture() API until it returns NV_ENC_SUCCESS.
The client must always lock the bitstream data in order in which it has submitted. This is true for both asynchronous and synchronous mode.

Picture type decision:

If the client is taking the picture type decision and it must disable the picture type decision module in NvEncodeAPI by setting NV_ENC_INITIALIZE_PARAMS::enablePTD to 0. In this case the client is required to send the picture in encoding order to NvEncodeAPI by doing the re-ordering for B frames.
If the client doesn't want to take the picture type decision it can enable picture type decision module in the NvEncodeAPI interface by setting NV_ENC_INITIALIZE_PARAMS::enablePTD to 1 and send the input pictures in display order.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***createEncodeParams*** Refer _NV_ENC_INITIALIZE_PARAMS for details.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_GENERIC

4.2.1.22 NVENCSTATUS NVENCAPI NvEncInvalidateRefFrames (void ∗ *encoder*, uint64_t *invalidRefFrameTimeStamp*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***invalidRefFrameTimeStamp*** Timestamp of the invalid reference frames which needs to be invalidated.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_GENERIC

4.2.1.23 NVENCSTATUS NVENCAPI NvEncLockBitstream (void ∗ *encoder*, NV_ENC_LOCK_BITSTREAM ∗ *lockBitstreamBufferParams*)

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

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***lockBitstreamBufferParams*** Pointer to the _NV_ENC_LOCK_BITSTREAM structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_LOCK_BUSY
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.24 NVENCSTATUS NVENCAPI NvEncLockInputBuffer (void ∗ *encoder*, NV_ENC_LOCK_INPUT_BUFFER ∗ *lockInputBufferParams*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***lockInputBufferParams*** Pointer to the _NV_ENC_LOCK_INPUT_BUFFER structure

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_LOCK_BUSY
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.25 NVENCSTATUS NVENCAPI NvEncMapInputResource (void ∗ *encoder*, NV_ENC_MAP_INPUT_RESOURCE ∗ *mapInputResParams*)

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

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

↔ ***mapInputResParams*** Pointer to the _NV_ENC_MAP_INPUT_RESOURCE structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_RESOURCE_NOT_REGISTERED
NV_ENC_ERR_MAP_FAILED
NV_ENC_ERR_GENERIC

4.2.1.26 NVENCSTATUS NVENCAPI NvEncodeAPICreateInstance (NV_ENCODE_API_FUNCTION_-LIST ∗ *functionList*)

Entry Point to the NvEncodeAPI interface.

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

Parameters:

→ ***functionList***

Returns:

NV_ENC_SUCCESS NV_ENC_ERR_INVALID_PTR

4.2.1.27 NVENCSTATUS NVENCAPI NvEncodeAPIGetMaxSupportedVersion (uint32_t ∗ *version*)

This function can be used by clients to determine if the driver supports the NvEncodeAPI header the application was compiled with.

Parameters:

→ ***version*** Pointer to the requested value. The 4 least significant bits in the returned indicate the minor version and the rest of the bits indicate the major version of the largest supported version.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR

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

Deprecated.

Returns:

NV_ENC_ERR_INVALID_CALL

4.2.1.29 NVENCSTATUS NVENCAPI NvEncOpenEncodeSessionEx (NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS ∗ *openSessionExParams*, void ∗∗ *encoder*)

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

Parameters:

← ***openSessionExParams*** Pointer to a NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS structure.

→ ***encoder*** Encode Session pointer to the NvEncodeAPI interface.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_NO_ENCODE_DEVICE
NV_ENC_ERR_UNSUPPORTED_DEVICE
NV_ENC_ERR_INVALID_DEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_GENERIC

4.2.1.30 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

4.2.1.31 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

4.2.1.32 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

4.2.1.33 NVENCSTATUS NVENCAPI NvEncRunMotionEstimationOnly (void ∗ *encoder*, NV_ENC_MEONLY_PARAMS ∗ *meOnlyParams*)

This function is used to submit the input frame and reference frame for motion estimation. The ME parameters are passed using ∗meOnlyParams which is a pointer to _NV_ENC_MEONLY_PARAMS structure. Client must lock NV_ENC_CREATE_MV_BUFFER::mvBuffer using NvEncLockBitstream() API to get the motion vector data. to get motion vector data.

Parameters:

← ***encoder*** Pointer to the NvEncodeAPI interface.

← ***meOnlyParams*** Pointer to the _NV_ENC_MEONLY_PARAMS structure.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_NEED_MORE_INPUT
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_GENERIC

4.2.1.34 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

4.2.1.35 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 NvEncEncode-Picture()

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

4.2.1.36 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 succuessfully 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

4.2.1.37 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

4.2.1.38 NVENCSTATUS NVENCAPI NvEncUnregisterResource (void ∗ *encoder*, NV_ENC_REGISTERED_PTR *registeredResource*)

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

Parameters:

← ***encoder*** Pointer to the NVEncodeAPI interface.

← ***registeredResource*** The registered resource pointer that was returned in NvEncRegisterResource.

Returns:

NV_ENC_SUCCESS
NV_ENC_ERR_INVALID_PTR
NV_ENC_ERR_INVALID_ENCODERDEVICE
NV_ENC_ERR_DEVICE_NOT_EXIST
NV_ENC_ERR_UNSUPPORTED_PARAM
NV_ENC_ERR_OUT_OF_MEMORY
NV_ENC_ERR_INVALID_VERSION
NV_ENC_ERR_INVALID_PARAM
NV_ENC_ERR_ENCODER_NOT_INITIALIZED
NV_ENC_ERR_RESOURCE_NOT_REGISTERED
NV_ENC_ERR_GENERIC
NV_ENC_ERR_UNIMPLEMENTED

Chapter 5

Data Structure Documentation

5.1 _NV_ENC_CODEC_CONFIG Struct Reference

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

5.1.1 Detailed Description

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

5.2 _NV_ENC_CONFIG Struct Reference

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

5.2.1 Detailed Description

Encoder configuration parameters to be set during initialization.

5.3 _NV_ENC_CONFIG_H264 Struct Reference

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

5.3.1 Detailed Description

H264 encoder configuration parameters

5.4 _NV_ENC_CONFIG_H264_VUI_PARAMETERS Struct Reference

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

5.4.1 Detailed Description

H264 Video Usability Info parameters

5.5 _NV_ENC_CONFIG_HEVC Struct Reference

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

5.5.1 Detailed Description

HEVC encoder configuration parameters to be set during initialization.

5.6 _NV_ENC_INITIALIZE_PARAMS Struct Reference

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

5.6.1 Detailed Description

Encode Session Initialization parameters.

5.7 _NV_ENC_LOCK_BITSTREAM Struct Reference

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

5.7.1 Detailed Description

Bitstream buffer lock parameters.

5.8 _NV_ENC_LOCK_INPUT_BUFFER Struct Reference

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

5.8.1 Detailed Description

Uncompressed Input Buffer lock parameters.

5.9 _NV_ENC_MAP_INPUT_RESOURCE Struct Reference

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

5.9.1 Detailed Description

Map an input resource to a Nvidia Encoder Input Buffer

5.10 _NV_ENC_MEONLY_PARAMS Struct Reference

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

5.10.1 Detailed Description

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

5.11 _NV_ENC_PIC_PARAMS Struct Reference

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

5.11.1 Detailed Description

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

5.12 _NV_ENC_PIC_PARAMS_H264 Struct Reference

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

5.12.1 Detailed Description

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

5.13 _NV_ENC_PIC_PARAMS_HEVC Struct Reference

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

5.13.1 Detailed Description

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

5.14 _NV_ENC_PRESET_CONFIG Struct Reference

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

5.14.1 Detailed Description

Encoder preset config

5.15 _NV_ENC_RECONFIGURE_PARAMS Struct Reference

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

5.15.1 Detailed Description

Encode Session Reconfigured parameters.

5.16 _NV_ENC_REGISTER_RESOURCE Struct Reference

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

5.16.1 Detailed Description

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

5.17 _NV_ENC_SEI_PAYLOAD Struct Reference

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

5.17.1 Detailed Description

User SEI message

5.18 _NV_ENC_SEQUENCE_PARAM_PAYLOAD Struct Reference

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

5.18.1 Detailed Description

Sequence and picture paramaters payload.

5.19 _NV_ENC_STAT Struct Reference

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

5.19.1 Detailed Description

Encode Stats structure.

5.20 _NVENC_EXTERNAL_ME_HINT Struct Reference

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

5.20.1 Detailed Description

External Motion Vector hint structure.

5.21 _NVENC_EXTERNAL_ME_HINT_COUNTS_PER_BLOCKTYPE Struct Reference

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

5.21.1 Detailed Description

External motion vector hint counts per block type.

5.22 _NVENC_RECT Struct Reference

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

5.22.1 Detailed Description

Defines a Rectangle. Used in NV_ENC_PREPROCESS_FRAME.

5.23 GUID Struct Reference

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

Data Fields

- uint32_t Data1
- uint16_t Data2
- uint16_t Data3
- uint8_t Data4 [8]

5.23.1 Detailed Description

Abstracts the GUID structure for non-windows platforms.

5.23.2 Field Documentation

5.23.2.1 uint32_t GUID::Data1

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

5.23.2.2 uint16_t GUID::Data2

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

5.23.2.3 uint16_t GUID::Data3

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

5.23.2.4 uint8_t GUID::Data4[8]

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

5.24 NV_ENC_CAPS_PARAM Struct Reference

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

Data Fields

- uint32_t version
- NV_ENC_CAPS capsToQuery
- uint32_t reserved [62]

5.24.1 Detailed Description

Input struct for querying Encoding capabilities.

5.24.2 Field Documentation

5.24.2.1 NV_ENC_CAPS NV_ENC_CAPS_PARAM::capsToQuery

[in]: Specifies the encode capability to be queried. Client should pass a member for NV_ENC_CAPS enum.

5.24.2.2 uint32_t NV_ENC_CAPS_PARAM::reserved[62]

[in]: Reserved and must be set to 0

5.24.2.3 uint32_t NV_ENC_CAPS_PARAM::version

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

5.25 NV_ENC_CODEC_PIC_PARAMS Union Reference

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

Data Fields

- NV_ENC_PIC_PARAMS_H264 h264PicParams
- NV_ENC_PIC_PARAMS_HEVC hevcPicParams
- uint32_t reserved [256]

5.25.1 Detailed Description

Codec specific per-picture encoding parameters.

5.25.2 Field Documentation

5.25.2.1 NV_ENC_PIC_PARAMS_H264 NV_ENC_CODEC_PIC_PARAMS::h264PicParams

[in]: H264 encode picture params.

5.25.2.2 NV_ENC_PIC_PARAMS_HEVC NV_ENC_CODEC_PIC_PARAMS::hevcPicParams

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

5.25.2.3 uint32_t NV_ENC_CODEC_PIC_PARAMS::reserved[256]

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

5.26 NV_ENC_CREATE_BITSTREAM_BUFFER Struct Reference

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

Data Fields

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

5.26.1 Detailed Description

Creation parameters for output bitstream buffer.

5.26.2 Field Documentation

5.26.2.1 NV_ENC_OUTPUT_PTR NV_ENC_CREATE_BITSTREAM_BUFFER::bitstreamBuffer

[out]: Pointer to the output bitstream buffer

5.26.2.2 void∗ NV_ENC_CREATE_BITSTREAM_BUFFER::bitstreamBufferPtr

[out]: Reserved and should not be used

5.26.2.3 NV_ENC_MEMORY_HEAP NV_ENC_CREATE_BITSTREAM_BUFFER::memoryHeap

[in]: Deprecated. Will be removed in sdk 8.0

5.26.2.4 uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::reserved

[in]: Reserved and must be set to 0

5.26.2.5 uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::reserved1[58]

[in]: Reserved and should be set to 0

5.26.2.6 void∗ NV_ENC_CREATE_BITSTREAM_BUFFER::reserved2[64]

[in]: Reserved and should be set to NULL

5.26.2.7 uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::size

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

5.26.2.8 uint32_t NV_ENC_CREATE_BITSTREAM_BUFFER::version

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

5.27 NV_ENC_CREATE_INPUT_BUFFER Struct Reference

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

Data Fields

- uint32_t version
- uint32_t width
- uint32_t height
- NV_ENC_MEMORY_HEAP memoryHeap
- NV_ENC_BUFFER_FORMAT bufferFmt
- uint32_t reserved
- NV_ENC_INPUT_PTR inputBuffer
- void ∗ pSysMemBuffer
- uint32_t reserved1 [57]
- void ∗ reserved2 [63]

5.27.1 Detailed Description

Creation parameters for input buffer.

5.27.2 Field Documentation

5.27.2.1 NV_ENC_BUFFER_FORMAT NV_ENC_CREATE_INPUT_BUFFER::bufferFmt

[in]: Input buffer format

5.27.2.2 uint32_t NV_ENC_CREATE_INPUT_BUFFER::height

[in]: Input buffer width

5.27.2.3 NV_ENC_INPUT_PTR NV_ENC_CREATE_INPUT_BUFFER::inputBuffer

[out]: Pointer to input buffer

5.27.2.4 NV_ENC_MEMORY_HEAP NV_ENC_CREATE_INPUT_BUFFER::memoryHeap

[in]: Deprecated. Will be removed in sdk 8.0

5.27.2.5 void∗ NV_ENC_CREATE_INPUT_BUFFER::pSysMemBuffer

[in]: Pointer to existing sysmem buffer

5.27.2.6 uint32_t NV_ENC_CREATE_INPUT_BUFFER::reserved

[in]: Reserved and must be set to 0

5.27.2.7 uint32_t NV_ENC_CREATE_INPUT_BUFFER::reserved1[57]

[in]: Reserved and must be set to 0

5.27.2.8 void∗ NV_ENC_CREATE_INPUT_BUFFER::reserved2[63]

[in]: Reserved and must be set to NULL

5.27.2.9 uint32_t NV_ENC_CREATE_INPUT_BUFFER::version

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

5.27.2.10 uint32_t NV_ENC_CREATE_INPUT_BUFFER::width

[in]: Input buffer width

5.28 NV_ENC_CREATE_MV_BUFFER Struct Reference

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

Data Fields

- uint32_t version
- NV_ENC_OUTPUT_PTR mvBuffer
- uint32_t reserved1 [255]
- void * reserved2 [63]

5.28.1 Detailed Description

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

5.28.2 Field Documentation

5.28.2.1 NV_ENC_OUTPUT_PTR NV_ENC_CREATE_MV_BUFFER::mvBuffer

[out]: Pointer to the output motion vector buffer

5.28.2.2 uint32_t NV_ENC_CREATE_MV_BUFFER::reserved1[255]

[in]: Reserved and should be set to 0

5.28.2.3 void* NV_ENC_CREATE_MV_BUFFER::reserved2[63]

[in]: Reserved and should be set to NULL

5.28.2.4 uint32_t NV_ENC_CREATE_MV_BUFFER::version

[in]: Struct version. Must be set to NV_ENC_CREATE_MV_BUFFER_VER

5.29 NV_ENC_EVENT_PARAMS Struct Reference

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

Data Fields

- uint32_t version
- uint32_t reserved
- void ∗ completionEvent
- uint32_t reserved1 [253]
- void ∗ reserved2 [64]

5.29.1 Detailed Description

Event registration/unregistration parameters.

5.29.2 Field Documentation

5.29.2.1 void∗ NV_ENC_EVENT_PARAMS::completionEvent

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

5.29.2.2 uint32_t NV_ENC_EVENT_PARAMS::reserved

[in]: Reserved and must be set to 0

5.29.2.3 uint32_t NV_ENC_EVENT_PARAMS::reserved1[253]

[in]: Reserved and must be set to 0

5.29.2.4 void∗ NV_ENC_EVENT_PARAMS::reserved2[64]

[in]: Reserved and must be set to NULL

5.29.2.5 uint32_t NV_ENC_EVENT_PARAMS::version

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

5.30 NV_ENC_H264_MV_DATA Struct Reference

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

Data Fields

- NV_ENC_MVECTOR mv [4]
- uint8_t mbType
- uint8_t partitionType
- uint16_t reserved

5.30.1 Detailed Description

Motion vector structure per macroblock for H264 motion estimation.

5.30.2 Field Documentation

5.30.2.1 uint8_t NV_ENC_H264_MV_DATA::mbType

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

5.30.2.2 NV_ENC_MVECTOR NV_ENC_H264_MV_DATA::mv[4]

up to 4 vectors for 8x8 partition

5.30.2.3 uint8_t NV_ENC_H264_MV_DATA::partitionType

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

5.30.2.4 uint16_t NV_ENC_H264_MV_DATA::reserved

reserved padding for alignment

5.31 NV_ENC_HEVC_MV_DATA Struct Reference

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

Data Fields

- NV_ENC_MVECTOR mv [4]
- uint8_t cuType
- uint8_t cuSize
- uint8_t partitionMode
- uint8_t lastCUInCTB

5.31.1 Detailed Description

Motion vector structure per CU for HEVC motion estimation.

5.31.2 Field Documentation

5.31.2.1 uint8_t NV_ENC_HEVC_MV_DATA::cuSize

0: 8x8, 1: 16x16, 2: 32x32, 3: 64x64

5.31.2.2 uint8_t NV_ENC_HEVC_MV_DATA::cuType

0 (I), 1(P), 2 (Skip)

5.31.2.3 uint8_t NV_ENC_HEVC_MV_DATA::lastCUInCTB

Marker to separate CUs in the current CTB from CUs in the next CTB

5.31.2.4 NV_ENC_MVECTOR NV_ENC_HEVC_MV_DATA::mv[4]

up to 4 vectors within a CU

5.31.2.5 uint8_t NV_ENC_HEVC_MV_DATA::partitionMode

The CU partition mode 0 (2Nx2N), 1 (2NxN), 2(Nx2N), 3 (NxN), 4 (2NxnU), 5 (2NxnD), 6(nLx2N), 7 (nRx2N)

5.32 NV_ENC_MVECTOR Struct Reference

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

Data Fields

- int16_t mvx
- int16_t mvy

5.32.1 Detailed Description

Structs needed for ME only mode.

5.32.2 Field Documentation

5.32.2.1 int16_t NV_ENC_MVECTOR::mvx

the x component of MV in qpel units

5.32.2.2 int16_t NV_ENC_MVECTOR::mvy

the y component of MV in qpel units

5.33 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]

5.33.1 Detailed Description

Encoder Session Creation parameters

5.33.2 Field Documentation

5.33.2.1 uint32_t NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::apiVersion

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

5.33.2.2 void∗ NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::device

[in]: Pointer to client device.

5.33.2.3 NV_ENC_DEVICE_TYPE NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::deviceType

[in]: Specified the device Type

5.33.2.4 void∗ NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved

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

5.33.2.5 uint32_t NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved1[253]

[in]: Reserved and must be set to 0

5.33.2.6 void∗ NV_ENC_OPEN_ENCODE_SESSION_EX_PARAMS::reserved2[64]

[in]: Reserved and must be set to NULL

5.33.2.7 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.

5.34 NV_ENC_QP Struct Reference

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

5.34.1 Detailed Description

QP value for frames

5.35 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 enableLookahead:1
- uint32_t disableIadapt:1
- uint32_t disableBadapt:1
- uint32_t enableTemporalAQ:1
- uint32_t zeroReorderDelay:1
- uint32_t enableNonRefP:1
- uint32_t reservedBitFields:21
- NV_ENC_QP minQP
- NV_ENC_QP maxQP
- NV_ENC_QP initialRCQP
- uint32_t temporallayerIdxMask
- uint8_t temporalLayerQP [8]
- uint16_t targetQuality
- uint16_t lookaheadDepth

5.35.1 Detailed Description

Rate Control Configuration Paramters

5.35.2 Field Documentation

5.35.2.1 uint32_t NV_ENC_RC_PARAMS::averageBitRate

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

5.35.2.2 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.

5.35.2.3 uint32_t NV_ENC_RC_PARAMS::disableBadapt

[in]: Set this to 1 to disable adaptive B-frame decision (only has an effect when lookahead is enabled)

5.35.2.4 uint32_t NV_ENC_RC_PARAMS::disableIadapt

[in]: Set this to 1 to disable adaptive I-frame insertion at scene cuts (only has an effect when lookahead is enabled)

5.35.2.5 uint32_t NV_ENC_RC_PARAMS::enableAQ

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

5.35.2.6 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 (Not Supported when AQ(Spatial/Temporal) is enabled)

5.35.2.7 uint32_t NV_ENC_RC_PARAMS::enableInitialRCQP

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

5.35.2.8 uint32_t NV_ENC_RC_PARAMS::enableLookahead

[in]: Set this to 1 to enable lookahead with depth <lookaheadDepth> (if lookahead is enabled, input frames must remain available to the encoder until encode completion)

5.35.2.9 uint32_t NV_ENC_RC_PARAMS::enableMaxQP

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

5.35.2.10 uint32_t NV_ENC_RC_PARAMS::enableMinQP

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

5.35.2.11 uint32_t NV_ENC_RC_PARAMS::enableNonRefP

[in]: Set this to 1 to enable automatic insertion of non-reference P-frames (no effect if enablePTD=0)

5.35.2.12 uint32_t NV_ENC_RC_PARAMS::enableTemporalAQ

[in]: Set this to 1 to enable temporal AQ for H.264

5.35.2.13 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.

5.35.2.14 uint16_t NV_ENC_RC_PARAMS::lookaheadDepth

[in]: Maximum depth of lookahead with range 0-32 (only used if enableLookahead=1)

5.35.2.15 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.

5.35.2.16 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.

5.35.2.17 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.

5.35.2.18 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.

5.35.2.19 uint32_t NV_ENC_RC_PARAMS::reservedBitFields

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

5.35.2.20 uint16_t NV_ENC_RC_PARAMS::targetQuality

[in]: Target CQ (Constant Quality) level for VBR mode (range 0-51 with 0-automatic)

5.35.2.21 uint32_t NV_ENC_RC_PARAMS::temporallayerIdxMask

[in]: Specifies the temporal layers (as a bitmask) whose QPs have changed. Valid max bitmask is [2^NV_ENC_-CAPS_NUM_MAX_TEMPORAL_LAYERS - 1]

5.35.2.22 uint8_t NV_ENC_RC_PARAMS::temporalLayerQP[8]

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

5.35.2.23 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.

5.35.2.24 uint32_t NV_ENC_RC_PARAMS::vbvInitialDelay

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

5.35.2.25 uint32_t NV_ENC_RC_PARAMS::zeroReorderDelay

[in]: Set this to 1 to indicate zero latency operation (no reordering delay, num_reorder_frames=0)

5.36 NV_ENCODE_API_FUNCTION_LIST Struct Reference

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

Data Fields

- uint32_t version
- uint32_t reserved
- PNVENCOPENENCODESESSION nvEncOpenEncodeSession
- PNVENCGETENCODEGUIDCOUNT nvEncGetEncodeGUIDCount
- PNVENCGETENCODEPRESETCOUNT nvEncGetEncodeProfileGUIDCount
- PNVENCGETENCODEPRESETGUIDS nvEncGetEncodeProfileGUIDs
- PNVENCGETENCODEGUIDS nvEncGetEncodeGUIDs
- PNVENCGETINPUTFORMATCOUNT nvEncGetInputFormatCount
- PNVENCGETINPUTFORMATS nvEncGetInputFormats
- PNVENCGETENCODECAPS nvEncGetEncodeCaps
- PNVENCGETENCODEPRESETCOUNT nvEncGetEncodePresetCount
- PNVENCGETENCODEPRESETGUIDS nvEncGetEncodePresetGUIDs
- PNVENCGETENCODEPRESETCONFIG nvEncGetEncodePresetConfig
- PNVENCINITIALIZEENCODER nvEncInitializeEncoder
- PNVENCCREATEINPUTBUFFER nvEncCreateInputBuffer
- PNVENCDESTROYINPUTBUFFER nvEncDestroyInputBuffer
- PNVENCCREATEBITSTREAMBUFFER nvEncCreateBitstreamBuffer
- PNVENCDESTROYBITSTREAMBUFFER nvEncDestroyBitstreamBuffer
- PNVENCENCODEPICTURE nvEncEncodePicture
- PNVENCLOCKBITSTREAM nvEncLockBitstream
- PNVENCUNLOCKBITSTREAM nvEncUnlockBitstream
- PNVENCLOCKINPUTBUFFER nvEncLockInputBuffer
- PNVENCUNLOCKINPUTBUFFER nvEncUnlockInputBuffer
- PNVENCGETENCODESTATS nvEncGetEncodeStats
- PNVENCGETSEQUENCEPARAMS nvEncGetSequenceParams
- PNVENCREGISTERASYNCEVENT nvEncRegisterAsyncEvent
- PNVENCUNREGISTERASYNCEVENT nvEncUnregisterAsyncEvent
- PNVENCMAPINPUTRESOURCE nvEncMapInputResource
- PNVENCUNMAPINPUTRESOURCE nvEncUnmapInputResource
- PNVENCDESTROYENCODER nvEncDestroyEncoder
- PNVENCINVALIDATEREFFRAMES nvEncInvalidateRefFrames
- PNVENCOPENENCODESESSIONEX nvEncOpenEncodeSessionEx
- PNVENCREGISTERRESOURCE nvEncRegisterResource
- PNVENCUNREGISTERRESOURCE nvEncUnregisterResource
- PNVENCRECONFIGUREENCODER nvEncReconfigureEncoder
- PNVENCCREATEMVBUFFER nvEncCreateMVBuffer
- PNVENCDESTROYMVBUFFER nvEncDestroyMVBuffer
- PNVENCRUNMOTIONESTIMATIONONLY nvEncRunMotionEstimationOnly
- void ∗ reserved2 [281]

5.36.1 Detailed Description

NV_ENCODE_API_FUNCTION_LIST

5.36.2 Field Documentation

5.36.2.1 PNVENCCREATEBITSTREAMBUFFER NV_ENCODE_API_FUNCTION_-LIST::nvEncCreateBitstreamBuffer

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

5.36.2.2 PNVENCCREATEINPUTBUFFER NV_ENCODE_API_FUNCTION_-LIST::nvEncCreateInputBuffer

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

5.36.2.3 PNVENCCREATEMVBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncCreateMVBuffer

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

5.36.2.4 PNVENCDESTROYBITSTREAMBUFFER NV_ENCODE_API_FUNCTION_-LIST::nvEncDestroyBitstreamBuffer

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

5.36.2.5 PNVENCDESTROYENCODER NV_ENCODE_API_FUNCTION_LIST::nvEncDestroyEncoder

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

5.36.2.6 PNVENCDESTROYINPUTBUFFER NV_ENCODE_API_FUNCTION_-LIST::nvEncDestroyInputBuffer

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

5.36.2.7 PNVENCDESTROYMVBUFFER NV_ENCODE_API_FUNCTION_-LIST::nvEncDestroyMVBuffer

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

5.36.2.8 PNVENCENCODEPICTURE NV_ENCODE_API_FUNCTION_LIST::nvEncEncodePicture

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

5.36.2.9 PNVENCGETENCODECAPS NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeCaps

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

5.36.2.10 PNVENCGETENCODEGUIDCOUNT NV_ENCODE_API_FUNCTION_-LIST::nvEncGetEncodeGUIDCount

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

5.36.2.11 PNVENCGETENCODEGUIDS NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeGUIDs

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

5.36.2.12 PNVENCGETENCODEPRESETCONFIG NV_ENCODE_API_FUNCTION_-LIST::nvEncGetEncodePresetConfig

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

5.36.2.13 PNVENCGETENCODEPRESETCOUNT NV_ENCODE_API_FUNCTION_-LIST::nvEncGetEncodePresetCount

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

5.36.2.14 PNVENCGETENCODEPRESETGUIDS NV_ENCODE_API_FUNCTION_-LIST::nvEncGetEncodePresetGUIDs

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

5.36.2.15 PNVENCGETENCODEPRESETCOUNT NV_ENCODE_API_FUNCTION_-LIST::nvEncGetEncodeProfileGUIDCount

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

5.36.2.16 PNVENCGETENCODEPRESETGUIDS NV_ENCODE_API_FUNCTION_-LIST::nvEncGetEncodeProfileGUIDs

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

5.36.2.17 PNVENCGETENCODESTATS NV_ENCODE_API_FUNCTION_LIST::nvEncGetEncodeStats

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

5.36.2.18 PNVENCGETINPUTFORMATCOUNT NV_ENCODE_API_FUNCTION_-LIST::nvEncGetInputFormatCount

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

5.36.2.19 PNVENCGETINPUTFORMATS NV_ENCODE_API_FUNCTION_-LIST::nvEncGetInputFormats

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

5.36.2.20 PNVENCGETSEQUENCEPARAMS NV_ENCODE_API_FUNCTION_-LIST::nvEncGetSequenceParams

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

5.36.2.21 PNVENCINITIALIZEENCODER NV_ENCODE_API_FUNCTION_-LIST::nvEncInitializeEncoder

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

5.36.2.22 PNVENCINVALIDATEREFFRAMES NV_ENCODE_API_FUNCTION_-LIST::nvEncInvalidateRefFrames

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

5.36.2.23 PNVENCLOCKBITSTREAM NV_ENCODE_API_FUNCTION_LIST::nvEncLockBitstream

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

5.36.2.24 PNVENCLOCKINPUTBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncLockInputBuffer

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

5.36.2.25 PNVENCMAPINPUTRESOURCE NV_ENCODE_API_FUNCTION_-LIST::nvEncMapInputResource

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

5.36.2.26 PNVENCOPENENCODESESSION NV_ENCODE_API_FUNCTION_-LIST::nvEncOpenEncodeSession

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

5.36.2.27 PNVENCOPENENCODESESSIONEX NV_ENCODE_API_FUNCTION_-LIST::nvEncOpenEncodeSessionEx

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

5.36.2.28 PNVENCRECONFIGUREENCODER NV_ENCODE_API_FUNCTION_-LIST::nvEncReconfigureEncoder

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

5.36.2.29 PNVENCREGISTERASYNCEVENT NV_ENCODE_API_FUNCTION_-LIST::nvEncRegisterAsyncEvent

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

5.36.2.30 PNVENCREGISTERRESOURCE NV_ENCODE_API_FUNCTION_-LIST::nvEncRegisterResource

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

5.36.2.31 PNVENCRUNMOTIONESTIMATIONONLY NV_ENCODE_API_FUNCTION_LIST::nvEncRunMotionEstimationOnly

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

5.36.2.32 PNVENCUNLOCKBITSTREAM NV_ENCODE_API_FUNCTION_LIST::nvEncUnlockBitstream

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

5.36.2.33 PNVENCUNLOCKINPUTBUFFER NV_ENCODE_API_FUNCTION_LIST::nvEncUnlockInputBuffer

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

5.36.2.34 PNVENCUNMAPINPUTRESOURCE NV_ENCODE_API_FUNCTION_LIST::nvEncUnmapInputResource

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

5.36.2.35 PNVENCUNREGISTERASYNCEVENT NV_ENCODE_API_FUNCTION_LIST::nvEncUnregisterAsyncEvent

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

5.36.2.36 PNVENCUNREGISTERRESOURCE NV_ENCODE_API_FUNCTION_LIST::nvEncUnregisterResource

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

5.36.2.37 uint32_t NV_ENCODE_API_FUNCTION_LIST::reserved

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

5.36.2.38 void∗ NV_ENCODE_API_FUNCTION_LIST::reserved2[281]

[in]: Reserved and must be set to NULL

5.36.2.39 uint32_t NV_ENCODE_API_FUNCTION_LIST::version

[in]: Client should pass NV_ENCODE_API_FUNCTION_LIST_VER.

Index