



MT DirectStream for VDI

API Reference

Version: v 1.0.0

2022-08-29

1 MT DirectStream for VDI	1
1.1 MTEncode	1
2 Module Index	2
2.1 Modules	2
3 Class Index	3
3.1 Class List	3
4 File Index	4
4.1 File List	4
5 Module Documentation	5
5.1 MTEncodeAPI data structures	5
5.1.1 Detailed Description	7
5.1.2 Macro Definition Documentation	7
5.1.2.1 MTENC_INFINITE_GOPLength	7
5.1.2.2 MTENCAPI_VERSION	7
5.1.3 Typedef Documentation	7
5.1.3.1 MT_ENC_BUFFER_FORMAT	7
5.1.3.2 MT_ENC_CODEC_CONFIG	8
5.1.3.3 MT_ENC_CODEC_ID	8
5.1.3.4 MT_ENC_CODEC_PROFILE_ID	8
5.1.3.5 MT_ENC_CONFIG	8
5.1.3.6 MT_ENC_CONFIG_H264	8
5.1.3.7 MT_ENC_CONFIG_HEVC	8
5.1.3.8 MT_ENC_CREATE_ENCODER_PARAMS	8
5.1.3.9 MT_ENC_CREATE_OUTPUT_BUFFER	8
5.1.3.10 MT_ENC_DEVICE_TYPE	9
5.1.3.11 MT_ENC_INIT_PARAMS	9
5.1.3.12 MT_ENC_INPUT_PTR	9
5.1.3.13 MT_ENC_INPUT_RESOURCE_TYPE	9
5.1.3.14 MT_ENC_LEVEL	9
5.1.3.15 MT_ENC_LOCK_BUFFER	9
5.1.3.16 MT_ENC_MAP_RESOURCE	9
5.1.3.17 MT_ENC_OUTPUT_PTR	9
5.1.3.18 MT_ENC_PARAMS_RC_MODE	10
5.1.3.19 MT_ENC_PIC_FLAGS	10
5.1.3.20 MT_ENC_PIC_PARAMS	10
5.1.3.21 MT_ENC_PIC_TYPE	10
5.1.3.22 MT_ENC_PRESET_CONFIG	10
5.1.3.23 MT_ENC_PRESET_ID	10
5.1.3.24 MT_ENC_QP	10
5.1.3.25 MT_ENC_RC_PARAMS	10

5.1.3.26 MT_ENCODE_API_FUNCTION_LIST	11
5.1.3.27 MT_ENCODER_ROI	11
5.1.3.28 MT_RECTANGLE	11
5.1.3.29 MTENCSTATUS	11
5.1.4 Enumeration Type Documentation	11
5.1.4.1 _MT_ENC_BUFFER_FORMAT	11
5.1.4.2 _MT_ENC_CODEC_ID	12
5.1.4.3 _MT_ENC_CODEC_PROFILE_ID	12
5.1.4.4 _MT_ENC_DEVICE_TYPE	12
5.1.4.5 _MT_ENC_INPUT_RESOURCE_TYPE	12
5.1.4.6 _MT_ENC_LEVEL	13
5.1.4.7 _MT_ENC_PARAMS_RC_MODE	13
5.1.4.8 _MT_ENC_PIC_FLAGS	13
5.1.4.9 _MT_ENC_PIC_TYPE	13
5.1.4.10 _MT_ENC_PRESET_ID	14
5.1.4.11 _MTENCSTATUS	14
5.2 MTEncodeAPI functions	15
5.2.1 Detailed Description	16
5.2.2 Function Documentation	16
5.2.2.1 MTEncCreateEncoder()	16
5.2.2.2 MTEncCreateOutputBuffer()	16
5.2.2.3 MTEncEncodeFrame()	17
5.2.2.4 MTEncGetPresetConfig()	18
5.2.2.5 MTEncInitEncoder()	18
5.2.2.6 MTEncLockOutputBuffer()	20
5.2.2.7 MTEncMapResource()	20
5.2.2.8 MTEncodeAPICreateInstance()	21
5.2.2.9 MTEncodeAPIGetMaxSupportedVersion()	21
5.2.2.10 MTEncReleaseEncoder()	22
5.2.2.11 MTEncReleaseOutputBuffer()	22
5.2.2.12 MTEncUnlockOutputBuffer()	23
5.2.2.13 MTEncUnmapResource()	23
6 Class Documentation	25
6.1 _MT_ENC_CODEC_CONFIG Struct Reference	25
6.1.1 Detailed Description	25
6.1.2 Member Data Documentation	25
6.1.2.1 h264Config	25
6.1.2.2 hevcConfig	25
6.1.2.3 reserved1	26
6.1.2.4 reserved2	26
6.2 _MT_ENC_CONFIG Struct Reference	26

6.2.1 Detailed Description	26
6.2.2 Member Data Documentation	26
6.2.2.1 bDepth	26
6.2.2.2 encodeCodecConfig	27
6.2.2.3 frameIntervalP	27
6.2.2.4 gopLength	27
6.2.2.5 profileID	27
6.2.2.6 rcParams	27
6.2.2.7 refs	27
6.2.2.8 reserved1	27
6.2.2.9 reserved2	28
6.2.2.10 version	28
6.3 _MT_ENC_CONFIG_H264 Struct Reference	28
6.3.1 Detailed Description	28
6.3.2 Member Data Documentation	28
6.3.2.1 idrPeriod	28
6.3.2.2 level	29
6.3.2.3 reserved1	29
6.3.2.4 reserved2	29
6.4 _MT_ENC_CONFIG_HEVC Struct Reference	29
6.4.1 Detailed Description	29
6.4.2 Member Data Documentation	29
6.4.2.1 idrPeriod	30
6.4.2.2 level	30
6.4.2.3 pixelBitDepthMinus8	30
6.4.2.4 reserved1	30
6.4.2.5 reserved2	30
6.4.2.6 reservedP1	30
6.4.2.7 reservedP2	30
6.4.2.8 tier	31
6.5 _MT_ENC_CREATE_ENCODER_PARAMS Struct Reference	31
6.5.1 Detailed Description	31
6.5.2 Member Data Documentation	31
6.5.2.1 device	31
6.5.2.2 deviceType	31
6.5.2.3 reserved1	32
6.5.2.4 reserved2	32
6.5.2.5 version	32
6.6 _MT_ENC_CREATE_OUTPUT_BUFFER Struct Reference	32
6.6.1 Detailed Description	32
6.6.2 Member Data Documentation	32
6.6.2.1 outputBuffer	33

6.6.2.2 reserved	33
6.6.2.3 reserved1	33
6.6.2.4 reserved2	33
6.6.2.5 version	33
6.7 _MT_ENC_INIT_PARAMS Struct Reference	33
6.7.1 Detailed Description	34
6.7.2 Member Data Documentation	34
6.7.2.1 enableEncodeAsync	34
6.7.2.2 encodeConfig	34
6.7.2.3 encodeHeight	34
6.7.2.4 encodeID	35
6.7.2.5 encodeWidth	35
6.7.2.6 frameRateDen	35
6.7.2.7 frameRateNum	35
6.7.2.8 maxEncodeHeight	35
6.7.2.9 maxEncodeWidth	35
6.7.2.10 presetID	35
6.7.2.11 reserved1	36
6.7.2.12 reserved2	36
6.7.2.13 version	36
6.8 _MT_ENC_LOCK_BUFFER Struct Reference	36
6.8.1 Detailed Description	36
6.8.2 Member Data Documentation	36
6.8.2.1 frameIdx	37
6.8.2.2 lockedOutputBuffer	37
6.8.2.3 outputBufferPtr	37
6.8.2.4 outputBufferSizeInBytes	37
6.8.2.5 pictureType	37
6.8.2.6 reserved1	37
6.8.2.7 reserved2	37
6.8.2.8 version	38
6.9 _MT_ENC_MAP_RESOURCE Struct Reference	38
6.9.1 Detailed Description	38
6.9.2 Member Data Documentation	38
6.9.2.1 mappedResource	38
6.9.2.2 reserved1	38
6.9.2.3 reserved2	39
6.9.2.4 resourceToMap	39
6.9.2.5 resourceType	39
6.9.2.6 version	39
6.10 _MT_ENC_PIC_PARAMS Struct Reference	39
6.10.1 Detailed Description	40

6.10.2 Member Data Documentation	40
6.10.2.1 bufferFmt	40
6.10.2.2 completionEvent	40
6.10.2.3 encodePicFlags	40
6.10.2.4 inputBuffer	40
6.10.2.5 inputHeight	40
6.10.2.6 inputPitch	40
6.10.2.7 inputWidth	41
6.10.2.8 outputBuffer	41
6.10.2.9 reserved0	41
6.10.2.10 reserved1	41
6.10.2.11 reserved2	41
6.10.2.12 roi	41
6.10.2.13 roiNumber	41
6.10.2.14 version	42
6.11 _MT_ENC_PRESET_CONFIG Struct Reference	42
6.11.1 Detailed Description	42
6.11.2 Member Data Documentation	42
6.11.2.1 presetCfg	42
6.11.2.2 reserved	42
6.11.2.3 reserved1	43
6.11.2.4 reserved2	43
6.11.2.5 version	43
6.12 _MT_ENC_QP Struct Reference	43
6.12.1 Detailed Description	43
6.12.2 Member Data Documentation	43
6.12.2.1 qpInterB	43
6.12.2.2 qpInterP	44
6.12.2.3 qpIntra	44
6.13 _MT_ENC_RC_PARAMS Struct Reference	44
6.13.1 Detailed Description	44
6.13.2 Member Data Documentation	44
6.13.2.1 averageBitRate	44
6.13.2.2 constQP	45
6.13.2.3 maxBitRate	45
6.13.2.4 rateControlMode	45
6.13.2.5 reserved1	45
6.13.2.6 reserved2	45
6.14 _MT_ENCODE_API_FUNCTION_LIST Struct Reference	45
6.14.1 Detailed Description	46
6.14.2 Member Data Documentation	46
6.14.2.1 mtEncCreateEncoder	46

6.14.2.2 mtEncCreateOutputBuffer	46
6.14.2.3 mtEncEncodeFrame	46
6.14.2.4 mtEncGetPresetConfig	47
6.14.2.5 mtEncInitEncoder	47
6.14.2.6 mtEncLockOutputBuffer	47
6.14.2.7 mtEncMapResource	47
6.14.2.8 mtEncReleaseEncoder	47
6.14.2.9 mtEncReleaseOutputBuffer	47
6.14.2.10 mtEncUnlockOutputBuffer	47
6.14.2.11 mtEncUnmapResource	48
6.14.2.12 reserved1	48
6.14.2.13 sdkVersion	48
6.14.2.14 version	48
6.15 _MT_ENCODER_ROI Struct Reference	48
6.15.1 Detailed Description	48
6.15.2 Member Data Documentation	49
6.15.2.1 roi_rectangle	49
6.15.2.2 roi_value	49
6.16 _MT_RECTANGLE Struct Reference	49
6.16.1 Detailed Description	49
7 File Documentation	50
7.1 mtEncodeAPI.h	50
Index	56

Chapter 1

MT DirectStream for VDI

1.1 MTEncode

Moore Threads GPUs contain a hardware-based H.264 video encoder (hereafter referred to as MTEncode). The MTEncode hardware takes YUV as input and generates an H.264-compliant video bitstream. The MTEncodeAPI functions, structures, and other parameters are exposed in [mtEncodeAPI.h](#). Broadly, the encoding flow consists of the following steps:

1. Initializes the encoder.
2. Sets up the desired encoding parameters.
3. Allocates input/output buffers.
4. Copies frames to the input buffers and reads bitstreams from the output buffers.
5. Cleans up - releases all allocated input/output buffers.
6. Releases the encoder.

```
// pseudocode
// open encoder
hModule = LoadLibrary(TEXT("mtEncodeAPI64.dll"));
pfCreateInstance = GetProcAddress(hModule, "MTEncodeAPICreateInstance");
MT_ENCODE_API_FUNCTION_LIST m_mtEnc = {};
pfCreateInstance(&m_mtEnc);
m_mtEnc.mtEncCreateEncoder(&createEncoderParams, &m_hEncoder)
// get preset configuration
MT_ENC_PRESET_CONFIG presetConfig = {};
m_mtEnc.mtEncGetPresetConfig(m_hEncoder, ...);
// configuration parameters here...
// init encoder
m_mtEnc.mtEncInitEncoder(m_hEncoder, ...)
// create input and output buffers
m_mtEnc.mtEncCreateOutputBuffer(m_hEncoder, ...)
while (1) {
    // fill data into an input buffer
    // encode frame
    m_mtEnc.mtEncEncodeFrame(m_hEncoder, ...)
    // get bitstream
    m_mtEnc.mtEncLockOutputBuffer(m_hEncoder, ...)
    m_mtEnc.mtEncUnlockOutputBuffer(m_hEncoder, ...)
}
// release resource & destroy encoder
m_mtEnc.mtEncReleaseOutputBuffer(m_hEncoder, ...)
m_mtEnc.mtEncReleaseEncoder(m_hEncoder)
FreeLibrary(hModule)
```

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

MTEncodeAPI data structures	5
MTEncodeAPI functions	15

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_MT_ENC_CODEC_CONFIG	25
_MT_ENC_CONFIG	26
_MT_ENC_CONFIG_H264	28
_MT_ENC_CONFIG_HEVC	29
_MT_ENC_CREATE_ENCODER_PARAMS	31
_MT_ENC_CREATE_OUTPUT_BUFFER	32
_MT_ENC_INIT_PARAMS	33
_MT_ENC_LOCK_BUFFER	36
_MT_ENC_MAP_RESOURCE	38
_MT_ENC_PIC_PARAMS	39
_MT_ENC_PRESET_CONFIG	42
_MT_ENC_QP	43
_MT_ENC_RC_PARAMS	44
_MT_ENCODE_API_FUNCTION_LIST	45
_MT_ENCODER_ROI	48
_MT_RECTANGLE	49

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

mtEncodeAPI.h	50
---	----

Chapter 5

Module Documentation

5.1 MTEncodeAPI data structures

Classes

- struct [_MT_ENC_CREATE_OUTPUT_BUFFER](#)
- struct [_MT_ENC_QP](#)
- struct [_MT_ENC_RC_PARAMS](#)
- struct [_MT_ENC_CONFIG_H264](#)
- struct [_MT_ENC_CONFIG_HEVC](#)
- struct [_MT_ENC_CODEC_CONFIG](#)
- struct [_MT_ENC_CONFIG](#)
- struct [_MT_ENC_INIT_PARAMS](#)
- struct [_MT_ENC_PRESET_CONFIG](#)
- struct [_MT_RECTANGLE](#)
- struct [_MT_ENCODER_ROI](#)
- struct [_MT_ENC_PIC_PARAMS](#)
- struct [_MT_ENC_LOCK_BUFFER](#)
- struct [_MT_ENC_MAP_RESOURCE](#)
- struct [_MT_ENC_CREATE_ENCODER_PARAMS](#)
- struct [_MT_ENCODE_API_FUNCTION_LIST](#)

Macros

- #define [MTENCAPI_VERSION](#) ((MTENCAPI_MAJOR_VERSION << 24) | (MTENCAPI_MINOR_VERSION << 16) | MTENCAPI_PATCH_VERSION)
- #define [MTENC_INFINITE_GOPLength](#) 0xFFFFFFFF

Typedefs

- typedef void * [MT_ENC_INPUT_PTR](#)
- typedef void * [MT_ENC_OUTPUT_PTR](#)
- typedef enum [_MT_ENC_CODEC_ID](#) [MT_ENC_CODEC_ID](#)
- typedef enum [_MT_ENC_CODEC_PROFILE_ID](#) [MT_ENC_CODEC_PROFILE_ID](#)
- typedef enum [_MT_ENC_PRESET_ID](#) [MT_ENC_PRESET_ID](#)
- typedef enum [_MT_ENC_PARAMS_RC_MODE](#) [MT_ENC_PARAMS_RC_MODE](#)
- typedef enum [_MT_ENC_PIC_TYPE](#) [MT_ENC_PIC_TYPE](#)
- typedef enum [_MT_ENC_BUFFER_FORMAT](#) [MT_ENC_BUFFER_FORMAT](#)
- typedef enum [_MT_ENC_LEVEL](#) [MT_ENC_LEVEL](#)
- typedef enum [_MTENCSTATUS](#) [MTENCSTATUS](#)
- typedef enum [_MT_ENC_PIC_FLAGS](#) [MT_ENC_PIC_FLAGS](#)
- typedef enum [_MT_ENC_INPUT_RESOURCE_TYPE](#) [MT_ENC_INPUT_RESOURCE_TYPE](#)
- typedef enum [_MT_ENC_DEVICE_TYPE](#) [MT_ENC_DEVICE_TYPE](#)
- typedef struct [_MT_ENC_CREATE_OUTPUT_BUFFER](#) [MT_ENC_CREATE_OUTPUT_BUFFER](#)
- typedef struct [_MT_ENC_QP](#) [MT_ENC_QP](#)
- typedef struct [_MT_ENC_RC_PARAMS](#) [MT_ENC_RC_PARAMS](#)
- typedef struct [_MT_ENC_CONFIG_H264](#) [MT_ENC_CONFIG_H264](#)
- typedef struct [_MT_ENC_CONFIG_HEVC](#) [MT_ENC_CONFIG_HEVC](#)
- typedef struct [_MT_ENC_CODEC_CONFIG](#) [MT_ENC_CODEC_CONFIG](#)
- typedef struct [_MT_ENC_CONFIG](#) [MT_ENC_CONFIG](#)
- typedef struct [_MT_ENC_INIT_PARAMS](#) [MT_ENC_INIT_PARAMS](#)
- typedef struct [_MT_ENC_PRESET_CONFIG](#) [MT_ENC_PRESET_CONFIG](#)
- typedef struct [_MT_RECTANGLE](#) [MT_RECTANGLE](#)
- typedef struct [_MT_ENCODER_ROI](#) [MT_ENCODER_ROI](#)
- typedef struct [_MT_ENC_PIC_PARAMS](#) [MT_ENC_PIC_PARAMS](#)
- typedef struct [_MT_ENC_LOCK_BUFFER](#) [MT_ENC_LOCK_BUFFER](#)
- typedef struct [_MT_ENC_MAP_RESOURCE](#) [MT_ENC_MAP_RESOURCE](#)
- typedef struct [_MT_ENC_CREATE_ENCODER_PARAMS](#) [MT_ENC_CREATE_ENCODER_PARAMS](#)
- typedef struct [_MT_ENCODE_API_FUNCTION_LIST](#) [MT_ENCODE_API_FUNCTION_LIST](#)

Enumerations

- enum [_MT_ENC_CODEC_ID](#) {
 [MT_ENC_CODEC_ID_H264](#) = 0x1 ,
 [MT_ENC_CODEC_ID_HEVC](#) = 0x2 }
- enum [_MT_ENC_CODEC_PROFILE_ID](#)
- enum [_MT_ENC_PRESET_ID](#) { [MT_ENC_PRESET_ID_DEFAULT](#) = 0x1 }
- enum [_MT_ENC_PARAMS_RC_MODE](#) {
 [MT_ENC_PARAMS_RC_CONSTQP](#) = 0x0 ,
 [MT_ENC_PARAMS_RC_CBR](#) = 0x1 ,
 [MT_ENC_PARAMS_RC_VBR](#) = 0x2 }
- enum [_MT_ENC_PIC_TYPE](#) {
 [MT_ENC_PIC_TYPE_I](#) = 0x0 ,
 [MT_ENC_PIC_TYPE_P](#) = 0x1 ,
 [MT_ENC_PIC_TYPE_B](#) = 0x2 ,
 [MT_ENC_PIC_TYPE_IDR](#) = 0x3 ,
 [MT_ENC_PIC_TYPE_UNKNOWN](#) = 0xFF }
- enum [_MT_ENC_BUFFER_FORMAT](#) {
 [MT_ENC_BUFFER_FORMAT_UNDEFINED](#) = 0x00000000 ,
 [MT_ENC_BUFFER_FORMAT_NV12](#) = 0x00000001 ,
 [MT_ENC_BUFFER_FORMAT_BGRA](#) = 0x10000000 ,
 [MT_ENC_BUFFER_FORMAT_P010_LE](#) = 0x00010000 }
- enum [_MT_ENC_LEVEL](#)

- enum `_MTENCSTATUS` {
 `MT_ENC_SUCCESS` ,
 `MT_ENC_ERR_NO_ENCODE_DEVICE` ,
 `MT_ENC_ERR_UNSUPPORTED_DEVICE` ,
 `MT_ENC_ERR_INVALID_PARAM` ,
 `MT_ENC_ERR_OUT_OF_MEMORY` ,
 `MT_ENC_ERR_ENCODER_NOT_INITIALIZED` ,
 `MT_ENC_ERR_INVALID_VERSION` ,
 `MT_ENC_ERR_MAP_FAILED` ,
 `MT_ENC_ERR_RESOURCE_NOT_MAPPED` ,
 `MT_ENC_ERR_NEED_MORE_INPUT` ,
 `MT_ENC_ERR_ENCODER_BUSY` ,
 `MT_ENC_ERR_GENERIC` }
- enum `_MT_ENC_PIC_FLAGS` {
 `MT_ENC_PIC_FLAG_FORCEIDR` = 0x2 ,
 `MT_ENC_PIC_FLAG_EOS` = 0x8 }
- enum `_MT_ENC_INPUT_RESOURCE_TYPE` { `MT_ENC_INPUT_RESOURCE_TYPE_DIRECTX` = 0x0 }
- enum `_MT_ENC_DEVICE_TYPE` { `MT_ENC_DEVICE_TYPE_DIRECTX` = 0x0 }

5.1.1 Detailed Description

5.1.2 Macro Definition Documentation

5.1.2.1 MTENC_INFINITE_GOPLength

```
#define MTENC_INFINITE_GOPLength 0xFFFFFFFF
```

An infinite GOP length.

5.1.2.2 MTENCAPI_VERSION

```
#define MTENCAPI_VERSION ((MTENCAPI_MAJOR_VERSION << 24) | (MTENCAPI_MINOR_VERSION << 16) |  
MTENCAPI_PATCH_VERSION)
```

The MTEncodeAPI version.

5.1.3 Typedef Documentation

5.1.3.1 MT_ENC_BUFFER_FORMAT

```
typedef enum _MT_ENC_BUFFER_FORMAT MT_ENC_BUFFER_FORMAT
```

Input buffer formats.

5.1.3.2 MT_ENC_CODEC_CONFIG

```
typedef struct _MT_ENC_CODEC_CONFIG MT_ENC_CODEC_CONFIG
```

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

5.1.3.3 MT_ENC_CODEC_ID

```
typedef enum _MT_ENC_CODEC_ID MT_ENC_CODEC_ID
```

Codec IDs supported by the MTEncodeAPI interface.

5.1.3.4 MT_ENC_CODEC_PROFILE_ID

```
typedef enum _MT_ENC_CODEC_PROFILE_ID MT_ENC_CODEC_PROFILE_ID
```

Profile IDs supported by the MTEncodeAPI interface.

5.1.3.5 MT_ENC_CONFIG

```
typedef struct _MT_ENC_CONFIG MT_ENC_CONFIG
```

Encoder configuration parameters to be set during initialization.

5.1.3.6 MT_ENC_CONFIG_H264

```
typedef struct _MT_ENC_CONFIG_H264 MT_ENC_CONFIG_H264
```

H.264 encoder configuration parameters.

5.1.3.7 MT_ENC_CONFIG_HEVC

```
typedef struct _MT_ENC_CONFIG_HEVC MT_ENC_CONFIG_HEVC
```

HEVC encoder configuration parameters.

5.1.3.8 MT_ENC_CREATE_ENCODER_PARAMS

```
typedef struct _MT_ENC_CREATE_ENCODER_PARAMS MT_ENC_CREATE_ENCODER_PARAMS
```

Encoder creation parameters.

5.1.3.9 MT_ENC_CREATE_OUTPUT_BUFFER

```
typedef struct _MT_ENC_CREATE_OUTPUT_BUFFER MT_ENC_CREATE_OUTPUT_BUFFER
```

Creation parameters for output bitstream buffers.

5.1.3.10 MT_ENC_DEVICE_TYPE

```
typedef enum _MT_ENC_DEVICE_TYPE MT_ENC_DEVICE_TYPE
```

Encoder device types.

5.1.3.11 MT_ENC_INIT_PARAMS

```
typedef struct _MT_ENC_INIT_PARAMS MT_ENC_INIT_PARAMS
```

Encoder initialization parameters.

5.1.3.12 MT_ENC_INPUT_PTR

```
typedef void* MT_ENC_INPUT_PTR
```

The MTEncodeAPI input buffer.

5.1.3.13 MT_ENC_INPUT_RESOURCE_TYPE

```
typedef enum _MT_ENC_INPUT_RESOURCE_TYPE MT_ENC_INPUT_RESOURCE_TYPE
```

Input resource types.

5.1.3.14 MT_ENC_LEVEL

```
typedef enum _MT_ENC_LEVEL MT_ENC_LEVEL
```

Encoding levels.

5.1.3.15 MT_ENC_LOCK_BUFFER

```
typedef struct _MT_ENC_LOCK_BUFFER MT_ENC_LOCK_BUFFER
```

Bitstream buffer lock parameters.

5.1.3.16 MT_ENC_MAP_RESOURCE

```
typedef struct _MT_ENC_MAP_RESOURCE MT_ENC_MAP_RESOURCE
```

Maps an input resource to the MT encoder input buffer.

5.1.3.17 MT_ENC_OUTPUT_PTR

```
typedef void* MT_ENC_OUTPUT_PTR
```

The MTEncodeAPI output buffer.

5.1.3.18 MT_ENC_PARAMS_RC_MODE

```
typedef enum _MT_ENC_PARAMS_RC_MODE MT_ENC_PARAMS_RC_MODE
```

Rate control (RC) modes.

5.1.3.19 MT_ENC_PIC_FLAGS

```
typedef enum _MT_ENC_PIC_FLAGS MT_ENC_PIC_FLAGS
```

Picture encoding flags.

5.1.3.20 MT_ENC_PIC_PARAMS

```
typedef struct _MT_ENC_PIC_PARAMS MT_ENC_PIC_PARAMS
```

Picture encoding parameters, sent on a frame-by-frame basis.

5.1.3.21 MT_ENC_PIC_TYPE

```
typedef enum _MT_ENC_PIC_TYPE MT_ENC_PIC_TYPE
```

Input picture types.

5.1.3.22 MT_ENC_PRESET_CONFIG

```
typedef struct _MT_ENC_PRESET_CONFIG MT_ENC_PRESET_CONFIG
```

The encoder preset configuration structure.

5.1.3.23 MT_ENC_PRESET_ID

```
typedef enum _MT_ENC_PRESET_ID MT_ENC_PRESET_ID
```

Preset IDs supported by the MTEncodeAPI interface.

5.1.3.24 MT_ENC_QP

```
typedef struct _MT_ENC_QP MT_ENC_QP
```

QP values for frames.

5.1.3.25 MT_ENC_RC_PARAMS

```
typedef struct _MT_ENC_RC_PARAMS MT_ENC_RC_PARAMS
```

Rate control configuration parameters.

5.1.3.26 MT_ENCODE_API_FUNCTION_LIST

```
typedef struct _MT_ENCODE_API_FUNCTION_LIST MT_ENCODE_API_FUNCTION_LIST
```

A list of pointers to the MTEncodeAPI functions.

5.1.3.27 MT_ENCODER_ROI

```
typedef struct _MT_ENCODER_ROI MT_ENCODER_ROI
```

Encoding region-of-interest (ROI).

The ROI set through this structure is applicable only to the current frame or field, so it must be sent every frame or field to be applied. The encoder will use the ROI information to adjust the QP values of the MB's that fall within the ROIs.

5.1.3.28 MT_RECTANGLE

```
typedef struct _MT_RECTANGLE MT_RECTANGLE
```

A rectangle.

5.1.3.29 MTENCSTATUS

```
typedef enum _MTENCSTATUS MTENCSTATUS
```

Error codes.

5.1.4 Enumeration Type Documentation

5.1.4.1 _MT_ENC_BUFFER_FORMAT

```
enum _MT_ENC_BUFFER_FORMAT
```

Input buffer formats.

Enumerator

MT_ENC_BUFFER_FORMAT_UNDEFINED	The undefined buffer format.
MT_ENC_BUFFER_FORMAT_NV12	Semi-planar YUV. The Y plane is followed by the interleaved UV plane.
MT_ENC_BUFFER_FORMAT_BGRA	32-bit BGRA that uses 8 bits for storing each channel.
MT_ENC_BUFFER_FORMAT_P010_LE	Like NV12, with 10 bits per pixel (bpp) per component, data in the high bits, zeros in the low bits, little-endian.

5.1.4.2 _MT_ENC_CODEC_ID

enum [_MT_ENC_CODEC_ID](#)

Codec IDs supported by the MTEncodeAPI interface.

Enumerator

MT_ENC_CODEC_ID_H264	The H.264 codec ID.
MT_ENC_CODEC_ID_HEVC	The HEVC codec ID. HEVC is not supported now.

5.1.4.3 _MT_ENC_CODEC_PROFILE_ID

enum [_MT_ENC_CODEC_PROFILE_ID](#)

Profile IDs supported by the MTEncodeAPI interface.

5.1.4.4 _MT_ENC_DEVICE_TYPE

enum [_MT_ENC_DEVICE_TYPE](#)

Encoder device types.

Enumerator

MT_ENC_DEVICE_TYPE_DIRECTX	The encoder device type is a DirectX device.
----------------------------	--

5.1.4.5 _MT_ENC_INPUT_RESOURCE_TYPE

enum [_MT_ENC_INPUT_RESOURCE_TYPE](#)

Input resource types.

Enumerator

MT_ENC_INPUT_RESOURCE_TYPE_DIRECTX	The input resource type is a DirectX surface.
------------------------------------	---

5.1.4.6 _MT_ENC_LEVEL

enum [_MT_ENC_LEVEL](#)

Encoding levels.

5.1.4.7 _MT_ENC_PARAMS_RC_MODE

enum [_MT_ENC_PARAMS_RC_MODE](#)

Rate control (RC) modes.

Enumerator

MT_ENC_PARAMS_RC_CONSTQP	The constant QP mode.
MT_ENC_PARAMS_RC_CBR	The constant bit rate (CBR) mode.
MT_ENC_PARAMS_RC_VBR	The variable bit rate (VBR) mode.

5.1.4.8 _MT_ENC_PIC_FLAGS

enum [_MT_ENC_PIC_FLAGS](#)

Picture encoding flags.

Enumerator

MT_ENC_PIC_FLAG_FORCEIDR	Encodes the current picture as an IDR picture. This flag is only valid when the picture type decision is taken by the encoder.
MT_ENC_PIC_FLAG_EOS	Indicates the end of the input stream.

5.1.4.9 _MT_ENC_PIC_TYPE

enum [_MT_ENC_PIC_TYPE](#)

Input picture types.

Enumerator

MT_ENC_PIC_TYPE_I	An intra-predicted picture.
MT_ENC_PIC_TYPE_P	A forward-predicted picture.
MT_ENC_PIC_TYPE_B	A bi-directionally predicted picture.
MT_ENC_PIC_TYPE_IDR	An instantaneous decoding refresh (IDR) picture.
MT_ENC_PIC_TYPE_UNKNOWN	An unknown type.

5.1.4.10 _MT_ENC_PRESET_ID

enum [_MT_ENC_PRESET_ID](#)

Preset IDs supported by the MTEncodeAPI interface.

Enumerator

MT_ENC_PRESET_ID_DEFAULT	The default preset ID.
--------------------------	------------------------

5.1.4.11 _MTENCSTATUS

enum [_MTENCSTATUS](#)

Error codes.

Enumerator

MT_ENC_SUCCESS	This indicates that API call returned with no errors.
MT_ENC_ERR_NO_ENCODE_DEVICE	Devices capable of encoding are not detected or invalid.
MT_ENC_ERR_UNSUPPORTED_DEVICE	The devices are not supported.
MT_ENC_ERR_INVALID_PARAM	At least one parameter value that is passed to the API is invalid.
MT_ENC_ERR_OUT_OF_MEMORY	Out of memory.
MT_ENC_ERR_ENCODER_NOT_INITIALIZED	The encoder has not been initialized with MTEncInitEncoder() or initialization failed. The client cannot allocate input or output buffers or do any encoding-related operation until successfully initializing the encoder.
MT_ENC_ERR_INVALID_VERSION	The client uses an invalid version.
MT_ENC_ERR_MAP_FAILED	MTEncMapResource() failed to map the input resource that the client provides.
MT_ENC_ERR_RESOURCE_NOT_MAPPED	The client is attempting to unmap a resource that has not been successfully mapped.
MT_ENC_ERR_NEED_MORE_INPUT	The encoder requires more input buffers to produce an output buffer. If this error is returned from MTEncEncodeFrame() , this is not a fatal error. If the client is encoding using B-frames, MTEncEncodeFrame() might be buffering the input frames for re-ordering. A client that operates in the synchronous mode cannot call MTEncLockOutputBuffer() on the output bitstream buffers if MTEncEncodeFrame() returns the MT_ENC_ERR_NEED_MORE_INPUT error code. The client must keep providing input frames until the encoder returns MT_ENC_SUCCESS . After receiving MT_ENC_SUCCESS , the client can call MTEncLockOutputBuffer() on the output buffers in the same order in which it has called MTEncEncodeFrame() .

Enumerator

MT_ENC_ERR_ENCODER_BUSY	The hardware encoder is busy encoding and cannot encode the input. The client should call MTEncEncodeFrame() again after a few milliseconds.
MT_ENC_ERR_GENERIC	An unknown internal error has occurred.

5.2 MTEncodeAPI functions

Functions

- **MTENCSTATUS** MTENCAPI [MTEncCreateEncoder](#) ([MT_ENC_CREATE_ENCODER_PARAMS](#) *create↔ EncoderParams, void **encoder)
Creates an encoder.
- **MTENCSTATUS** MTENCAPI [MTEncGetPresetConfig](#) (void *encoder, [MT_ENC_CODEC_ID](#) encodeID, [MT_ENC_PRESET_ID](#) presetID, [MT_ENC_PRESET_CONFIG](#) *presetConfig)
Returns a preset configuration structure supported for a given preset ID.
- **MTENCSTATUS** MTENCAPI [MTEncInitEncoder](#) (void *encoder, [MT_ENC_INIT_PARAMS](#) *initParams)
Initializes the encoder.
- **MTENCSTATUS** MTENCAPI [MTEncCreateOutputBuffer](#) (void *encoder, [MT_ENC_CREATE_OUTPUT_BUFFER](#) *createOutputBufferParams)
Allocates an output bitstream buffer.
- **MTENCSTATUS** MTENCAPI [MTEncReleaseOutputBuffer](#) (void *encoder, [MT_ENC_OUTPUT_PTR](#) outputBuffer)
Releases an output buffer.
- **MTENCSTATUS** MTENCAPI [MTEncEncodeFrame](#) (void *encoder, [MT_ENC_PIC_PARAMS](#) *encodePic↔ Params)
Submits an input picture for encoding.
- **MTENCSTATUS** MTENCAPI [MTEncLockOutputBuffer](#) (void *encoder, [MT_ENC_LOCK_BUFFER](#) *lock↔ OutputBufferParams)
Locks output buffer.
- **MTENCSTATUS** MTENCAPI [MTEncUnlockOutputBuffer](#) (void *encoder, [MT_ENC_OUTPUT_PTR](#) output↔ Buffer)
Unlocks the output buffer.
- **MTENCSTATUS** MTENCAPI [MTEncMapResource](#) (void *encoder, [MT_ENC_MAP_RESOURCE](#) *map↔ InputResParams)
Maps an externally created input resource pointer for encoding.
- **MTENCSTATUS** MTENCAPI [MTEncUnmapResource](#) (void *encoder, [MT_ENC_INPUT_PTR](#) mapped↔ InputBuffer)
Unmaps an MT_ENC_INPUT_PTR which was mapped for encoding.
- **MTENCSTATUS** MTENCAPI [MTEncReleaseEncoder](#) (void *encoder)
Releases an encoder.
- **MTENCSTATUS** MTENCAPI [MTEncodeAPIGetMaxSupportedVersion](#) (uint32_t *version)
Gets the latest MTEncodeAPI version supported by the driver.
- **MTENCSTATUS** MTENCAPI [MTEncodeAPICreateInstance](#) ([MT_ENCODE_API_FUNCTION_LIST](#) *functionList)

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 MTEncCreateEncoder()

```
MTENCSTATUS MTENCAPI MTEncCreateEncoder (
    MT_ENC_CREATE_ENCODER_PARAMS * createEncoderParams,
    void ** encoder )
```

Creates an encoder.

Creates an encoder and returns a pointer to the encoder interface in the `**encoder` parameter. The client should start the encoding process by calling this function first. The client must pass a pointer to the DirectX device in the `*device` parameter. If the creation of encoder fails, the client must call [MTEncReleaseEncoder\(\)](#) before exiting.

Parameters

in	<code>createEncoderParams</code>	A pointer to MT_ENC_CREATE_ENCODER_PARAMS .
out	<code>encoder</code>	The encoder pointer to the MTEncodeAPI interface.

Returns

```
MT_ENC_SUCCESS
MT_ENC_ERR_INVALID_PARAM
MT_ENC_ERR_NO_ENCODE_DEVICE
MT_ENC_ERR_UNSUPPORTED_DEVICE
MT_ENC_ERR_GENERIC
```

5.2.2.2 MTEncCreateOutputBuffer()

```
MTENCSTATUS MTENCAPI MTEncCreateOutputBuffer (
    void * encoder,
    MT_ENC_CREATE_OUTPUT_BUFFER * createOutputBufferParams )
```

Allocates an output bitstream buffer.

This function is used to allocate an output bitstream buffer and returns a `MT_ENC_OUTPUT_PTR` to bitstream buffer to the client in [MT_ENC_CREATE_OUTPUT_BUFFER::outputBuffer](#). The client can only call this function after the encoder has been initialized using [MTEncInitEncoder\(\)](#). The minimum number of output buffers allocated by the client must be at least $(2 * \text{_MT_ENC_CONFIG::frameIntervalP} + 1)$. The client can only access the output bitstream data by locking `outputBuffer` using [MTEncLockOutputBuffer\(\)](#).

Parameters

in	<code>encoder</code>	A pointer to the MTEncodeAPI interface.
in, out	<code>createOutputBufferParams</code>	A pointer to MT_ENC_CREATE_OUTPUT_BUFFER .

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_INVALID_VERSION](#)
[MT_ENC_ERR_ENCODER_NOT_INITIALIZED](#)
[MT_ENC_ERR_GENERIC](#)

5.2.2.3 MTEncEncodeFrame()

```
MTENCSTATUS MTENCAPI MTEncEncodeFrame (  
    void * encoder,  
    MT\_ENC\_PIC\_PARAMS * encodePicParams )
```

Submits an input picture for encoding.

This function is used to submit an input picture buffer for encoding. The encoding parameters are passed using **encodePicParams* which is a pointer to [_MT_ENC_PIC_PARAMS](#).

The MTEncodeAPI interface might return the [MT_ENC_ERR_NEED_MORE_INPUT](#) error code but the client must not treat it as a fatal error. The MTEncodeAPI interface might not be able to submit an input picture buffer for encoding immediately due to re-ordering for B-frames. The MTEncodeAPI interface cannot submit the input picture which is decided to be encoded as a B-frame as it waits for backward reference from temporally subsequent frames. This input picture is buffered internally and waits for more input pictures to arrive. The client must not call [MTEncLockOutputBuffer\(\)](#) on the output buffers whose [MTEncEncodeFrame\(\)](#) returns [MT_ENC_ERR_NEED_MORE_INPUT](#). The client must wait for the MTEncodeAPI interface to return [MT_ENC_SUCCESS](#) before locking the output buffer to read the encoded bitstream data.

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in, out	<i>encodePicParams</i>	A pointer to the _MT_ENC_PIC_PARAMS structure.

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_INVALID_VERSION](#)
[MT_ENC_ERR_ENCODER_BUSY](#)
[MT_ENC_ERR_NEED_MORE_INPUT](#)
[MT_ENC_ERR_ENCODER_NOT_INITIALIZED](#)
[MT_ENC_ERR_GENERIC](#)

5.2.2.4 MTEncGetPresetConfig()

```
MTENCSTATUS MTENCAPI MTEncGetPresetConfig (
    void * encoder,
    MT_ENC_CODEC_ID encodeID,
    MT_ENC_PRESET_ID presetID,
    MT_ENC_PRESET_CONFIG * presetConfig )
```

Returns a preset configuration structure supported for a given preset ID.

The preset configuration structure can be modified by the client depending upon its use case and can be then used to initialize the encoder using [MTEncInitEncoder\(\)](#).

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in	<i>encodeID</i>	The encode ID, corresponding to which the supported presets are to be retrieved.
in	<i>presetID</i>	The preset ID, corresponding to which the encoding configurations are to be retrieved.
out	<i>presetConfig</i>	The requested preset encoder attribute set. For more information, see _MT_ENC_CONFIG .

Returns

```
MT_ENC_SUCCESS
MT_ENC_ERR_INVALID_PARAM
MT_ENC_ERR_NO_ENCODE_DEVICE
MT_ENC_ERR_OUT_OF_MEMORY
MT_ENC_ERR_INVALID_VERSION
MT_ENC_ERR_GENERIC
```

5.2.2.5 MTEncInitEncoder()

```
MTENCSTATUS MTENCAPI MTEncInitEncoder (
    void * encoder,
    MT_ENC_INIT_PARAMS * initParams )
```

Initializes the encoder.

This function must be used to initialize the encoder. The initialization parameters are passed using **initParams*. The client must send the following fields of the [_MT_ENC_INIT_PARAMS](#) structure with a valid value.

- [MT_ENC_INIT_PARAMS::encodeID](#)
- [MT_ENC_INIT_PARAMS::encodeWidth](#)
- [MT_ENC_INIT_PARAMS::encodeHeight](#)

The client can pass a preset ID directly to the MTEncodeAPI interface using [MT_ENC_INIT_PARAMS::presetID](#). If the client does not pass [MT_ENC_INIT_PARAMS::encodeConfig](#), the codec-specific parameters will be selected based on the preset ID. If the client passes a custom [_MT_ENC_CONFIG](#) structure through [MT_ENC_INIT_PARAMS::encodeConfig](#), it will override the codec-specific parameters based on the preset ID. It is recommended that even if the client passes a custom configuration, it should also send a preset ID. In this case, the preset ID passed by the client will not override any of the custom configuration parameters programmed by the client, it is only used as a hint by the MTEncodeAPI interface to determine certain encoder parameters which are not exposed to the client.

The encoder supports the following operation modes:

- The asynchronous mode
- The synchronous mode

The client can select asynchronous or synchronous mode by setting `enableEncodeAsync` in [_MT_ENC_INIT_PARAMS](#) to 1 or 0 respectively.

Asynchronous mode of operation

The asynchronous mode can be enabled by setting [MT_ENC_INIT_PARAMS::enableEncodeAsync](#) to 1. The client operating in the asynchronous mode must allocate a completion event object for each output buffer and pass the completion event object in the [MTEncEncodeFrame\(\)](#) function. The client can create another thread and wait on the event object to be signalled by the MTEncodeAPI 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 MTEncodeAPI interfaces to copy the bitstream data using the [MTEncLockOutputBuffer\(\)](#) function. This is the preferred mode of operation.

Synchronous mode of operation

The client can select the synchronous mode by setting [MT_ENC_INIT_PARAMS::enableEncodeAsync](#) to 0. The client working in the synchronous mode can work in a single-threaded or multithreaded mode. The client does not need to allocate any event objects. The client can only lock the bitstream data after the MTEncodeAPI interface has returned [MT_ENC_SUCCESS](#) from encoding pictures. MTEncodeAPI can return the [MT_ENC_ERR_NEED_MORE_INPUT](#) error code from [MTEncEncodeFrame\(\)](#). The client must not lock the output buffer in such a case but should send the next frame for encoding. The client must keep on calling [MTEncEncodeFrame\(\)](#) until it returns [MT_ENC_SUCCESS](#).

The client must always lock the bitstream data in the order in which it has been submitted.

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in	<i>initParams</i>	For more information, see _MT_ENC_INIT_PARAMS .

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_INVALID_VERSION](#)
[MT_ENC_ERR_GENERIC](#)

5.2.2.6 MTEncLockOutputBuffer()

```
MTENCSTATUS MTENCAPI MTEncLockOutputBuffer (
    void * encoder,
    MT_ENC_LOCK_BUFFER * lockOutputBufferParams )
```

Locks output buffer.

This function is used to lock the buffer to read the encoded data. The client can only access the encoded data by calling this function. A pointer to client accessible encoded data is returned in the [MT_ENC_LOCK_BUFFER::outputBufferPtr](#) field. The size of the encoded data in the output buffer is returned in [MT_ENC_LOCK_BUFFER::outputBufferSizeInBytes](#). The MTEncodeAPI interface also returns the output picture type of the encoded frame in [MT_ENC_LOCK_BUFFER::pictureType](#).

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in, out	<i>lockOutputBufferParams</i>	A pointer to the _MT_ENC_LOCK_BUFFER structure.

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_INVALID_VERSION](#)
[MT_ENC_ERR_ENCODER_NOT_INITIALIZED](#)
[MT_ENC_ERR_GENERIC](#)

5.2.2.7 MTEncMapResource()

```
MTENCSTATUS MTENCAPI MTEncMapResource (
    void * encoder,
    MT_ENC_MAP_RESOURCE * mapInputResParams )
```

Maps an externally created input resource pointer for encoding.

Maps an externally allocated input resource using and returns an [MT_ENC_INPUT_PTR](#) which can be used for encoding in [MTEncEncodeFrame\(\)](#). The mapped resource is returned in [MT_ENC_MAP_RESOURCE::mappedResource](#). This function provides synchronization guarantee that any graphics or compute 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 being mapped by the encoder.

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in, out	<i>mapInputResParams</i>	A pointer to _MT_ENC_MAP_RESOURCE .

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_INVALID_VERSION](#)
[MT_ENC_ERR_ENCODER_NOT_INITIALIZED](#)
[MT_ENC_ERR_MAP_FAILED](#)
[MT_ENC_ERR_GENERIC](#)

5.2.2.8 MTEncodeAPICreateInstance()

```
MTENCSTATUS MTENCAPI MTEncodeAPICreateInstance (
    MT_ENCODE_API_FUNCTION_LIST * functionList )
```

The entry point to the MTEncodeAPI interface.

Creates an instance of the MTEncodeAPI interface, and populates the `functionList` with function pointers to the API routines implemented by MTEncodeAPI.

Parameters

out	<i>functionList</i>	
-----	---------------------	--

Returns

[MT_ENC_SUCCESS](#) [MT_ENC_ERR_INVALID_PARAM](#) [MT_ENC_ERR_INVALID_VERSION](#)

5.2.2.9 MTEncodeAPIGetMaxSupportedVersion()

```
MTENCSTATUS MTENCAPI MTEncodeAPIGetMaxSupportedVersion (
    uint32_t * version )
```

Gets the latest MTEncodeAPI version supported by the driver.

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

Parameters

out	<i>version</i>	A pointer to the requested value. The 4 least significant bits in the returned value indicate the minor version and the rest of the bits indicate the major version of the latest supported version.
-----	----------------	--

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)

5.2.2.10 MTEncReleaseEncoder()

```
MTENCSTATUS MTENCAPI MTEncReleaseEncoder (
    void * encoder )
```

Releases an encoder.

Releases the encoder previously created using [MTEncCreateEncoder\(\)](#). The client must flush the encoder before freeing any resources. To flush the encoder, the client must pass a NULL encode picture packet and either wait for [MTEncEncodeFrame\(\)](#) to return in the synchronous mode. The client must free all the input and output resources created using MTEncodeAPI before releasing the encoder.

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
----	----------------	---

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_GENERIC](#)

5.2.2.11 MTEncReleaseOutputBuffer()

```
MTENCSTATUS MTENCAPI MTEncReleaseOutputBuffer (
    void * encoder,
    MT_ENC_OUTPUT_PTR outputBuffer )
```

Releases an output buffer.

This function is used to release the output bitstream buffer allocated using [MTEncCreateOutputBuffer\(\)](#). The client must release the output bitstream buffer using this function before destroying the encoder.

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in	<i>outputBuffer</i>	A pointer to the bitstream buffer being released.

Returns

MT_ENC_SUCCESS
MT_ENC_ERR_INVALID_PARAM
MT_ENC_ERR_NO_ENCODE_DEVICE
MT_ENC_ERR_OUT_OF_MEMORY
MT_ENC_ERR_INVALID_VERSION
MT_ENC_ERR_ENCODER_NOT_INITIALIZED
MT_ENC_ERR_GENERIC

5.2.2.12 MTEncUnlockOutputBuffer()

```
MTENCSTATUS MTENCAPI MTEncUnlockOutputBuffer (
    void * encoder,
    MT_ENC_OUTPUT_PTR outputBuffer )
```

Unlocks the output buffer.

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

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in, out	<i>outputBuffer</i>	The bitstream buffer pointer that is being unlocked.

Returns

MT_ENC_SUCCESS
MT_ENC_ERR_INVALID_PARAM
MT_ENC_ERR_NO_ENCODE_DEVICE
MT_ENC_ERR_OUT_OF_MEMORY
MT_ENC_ERR_ENCODER_NOT_INITIALIZED
MT_ENC_ERR_GENERIC

5.2.2.13 MTEncUnmapResource()

```
MTENCSTATUS MTENCAPI MTEncUnmapResource (
    void * encoder,
    MT_ENC_INPUT_PTR mappedInputBuffer )
```

Unmaps an MT_ENC_INPUT_PTR which was mapped for encoding.

Unmaps an input buffer that was previously mapped using [MTEncMapResource\(\)](#). The mapping created using [MTEncMapResource\(\)](#) should be invalidated using this function before the external resource is destroyed by the client. The client must unmap the buffer after [MTEncLockOutputBuffer\(\)](#) returns successfully for encode work submitted using the mapped input buffer.

Parameters

in	<i>encoder</i>	A pointer to the MTEncodeAPI interface.
in	<i>mappedInputBuffer</i>	A pointer to MT_ENC_INPUT_PTR .

Returns

[MT_ENC_SUCCESS](#)
[MT_ENC_ERR_INVALID_PARAM](#)
[MT_ENC_ERR_NO_ENCODE_DEVICE](#)
[MT_ENC_ERR_OUT_OF_MEMORY](#)
[MT_ENC_ERR_INVALID_VERSION](#)
[MT_ENC_ERR_ENCODER_NOT_INITIALIZED](#)
[MT_ENC_ERR_RESOURCE_NOT_MAPPED](#)
[MT_ENC_ERR_GENERIC](#)

Chapter 6

Class Documentation

6.1 `_MT_ENC_CODEC_CONFIG` Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- [MT_ENC_CONFIG_H264](#) `h264Config`
- [MT_ENC_CONFIG_HEVC](#) `hevcConfig`
- `uint32_t` [reserved1](#) [1280]
- `void *` [reserved2](#) [320]

6.1.1 Detailed Description

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

6.1.2 Member Data Documentation

6.1.2.1 `h264Config`

[MT_ENC_CONFIG_H264](#) `_MT_ENC_CODEC_CONFIG::h264Config`

[in]: Specifies the H.264-specific encoder configuration.

6.1.2.2 `hevcConfig`

[MT_ENC_CONFIG_HEVC](#) `_MT_ENC_CODEC_CONFIG::hevcConfig`

[in]: Specifies the HEVC-specific encoder configuration.

6.1.2.3 reserved1

```
uint32_t _MT_ENC_CODEC_CONFIG::reserved1[1280]
```

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

6.1.2.4 reserved2

```
void* _MT_ENC_CODEC_CONFIG::reserved2[320]
```

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

The documentation for this struct was generated from the following file:

- mtEncodeAPI.h

6.2 _MT_ENC_CONFIG Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- uint32_t [version](#)
- [MT_ENC_CODEC_PROFILE_ID](#) [profileID](#)
- uint32_t [gopLength](#)
- int32_t [frameIntervalP](#)
- [MT_ENC_RC_PARAMS](#) [rcParams](#)
- [MT_ENC_CODEC_CONFIG](#) [encodeCodecConfig](#)
- uint32_t [bDepth](#)
- uint32_t [refs](#)
- uint32_t [reserved1](#) [250]
- void * [reserved2](#) [64]

6.2.1 Detailed Description

Encoder configuration parameters to be set during initialization.

6.2.2 Member Data Documentation

6.2.2.1 bDepth

```
uint32_t _MT_ENC_CONFIG::bDepth
```

[in]: Not supported now. Specifies the maximum depth of hierarchical B-frames, which ranges from 1 to 3. The default value is 1. When `bDepth` is greater than 1, the number of successive B-frames should be 3 to 7.

6.2.2.2 encodeCodecConfig

`MT_ENC_CODEC_CONFIG` `_MT_ENC_CONFIG::encodeCodecConfig`

[in]: Specifies the codec-specific configuration parameters by using this union.

6.2.2.3 frameIntervalP

`int32_t` `_MT_ENC_CONFIG::frameIntervalP`

[in]: Specifies the GOP pattern as follows: `frameIntervalP == 0`: I, 1: IPP, 2: IBP, 3: IBBP. If `gopLength` is set to `MTENC_INFINITE_GOPLength`, `frameIntervalP` should be set to 1.

6.2.2.4 gopLength

`uint32_t` `_MT_ENC_CONFIG::gopLength`

[in]: Specifies the number of pictures in one GOP. A low-latency application client can set `gopLength` to `MTENC_INFINITE_GOPLength` so that keyframes are not inserted automatically.

6.2.2.5 profileID

`MT_ENC_CODEC_PROFILE_ID` `_MT_ENC_CONFIG::profileID`

[in]: Specifies the codec profile ID. If the client specifies `MT_ENC_CODEC_PROFILE_ID_AUTOSELECT`, the `MTEncodeAPI` interface will select the appropriate codec profile.

6.2.2.6 rcParams

`MT_ENC_RC_PARAMS` `_MT_ENC_CONFIG::rcParams`

[in]: Specifies the rate control parameters for the current encoder.

6.2.2.7 refs

`uint32_t` `_MT_ENC_CONFIG::refs`

[in]: Not supported now. Specifies the reference number of the P-frame, which ranges from 1 to 2. The default value is 1.

6.2.2.8 reserved1

`uint32_t` `_MT_ENC_CONFIG::reserved1[250]`

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

6.2.2.9 reserved2

```
void* _MT_ENC_CONFIG::reserved2[64]
```

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

6.2.2.10 version

```
uint32_t _MT_ENC_CONFIG::version
```

[in]: The struct version. Must be set to [MTENCAP_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.3 _MT_ENC_CONFIG_H264 Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- uint32_t [level](#)
- uint32_t [idrPeriod](#)
- uint32_t [reserved1](#) [254]
- void * [reserved2](#) [64]

6.3.1 Detailed Description

H.264 encoder configuration parameters.

6.3.2 Member Data Documentation

6.3.2.1 idrPeriod

```
uint32_t _MT_ENC_CONFIG_H264::idrPeriod
```

[in]: Specifies the IDR interval. If not set, this is made equal to [MT_ENC_CONFIG::gopLength](#). A low-latency application client can set the IDR interval to [MTENC_INFINITE_GOPELENGTH](#) so that IDR frames are not inserted automatically.

6.3.2.2 level

```
uint32_t _MT_ENC_CONFIG_H264::level
```

[in]: Specifies the encoding level. The client is recommended to set this to `MT_ENC_LEVEL_AUTOSELECT` to enable the MTEncodeAPI interface to select the correct level.

6.3.2.3 reserved1

```
uint32_t _MT_ENC_CONFIG_H264::reserved1[254]
```

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

6.3.2.4 reserved2

```
void* _MT_ENC_CONFIG_H264::reserved2[64]
```

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

The documentation for this struct was generated from the following file:

- `mtEncodeAPI.h`

6.4 _MT_ENC_CONFIG_HEVC Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- `uint32_t level`
- `uint32_t tier`
- `uint32_t idrPeriod`
- `uint32_t reservedP1: 11`
- `uint32_t pixelBitDepthMinus8: 3`
- `uint32_t reservedP2: 18`
- `uint32_t reserved1 [252]`
- `void * reserved2 [64]`

6.4.1 Detailed Description

HEVC encoder configuration parameters.

6.4.2 Member Data Documentation

6.4.2.1 idrPeriod

```
uint32_t _MT_ENC_CONFIG_HEVC::idrPeriod
```

[in]: Specifies the IDR interval. If not set, this is made equal to [MT_ENC_CONFIG::gopLength](#). A low-latency application client can set the IDR interval to [MTENC_INFINITE_GOPELENGTH](#) so that IDR frames are not inserted automatically.

6.4.2.2 level

```
uint32_t _MT_ENC_CONFIG_HEVC::level
```

[in]: Specifies the encoding level. The client is recommended to set this to [MT_ENC_LEVEL_AUTOSELECT](#) to enable the MTEncodeAPI interface to select the correct level.

6.4.2.3 pixelBitDepthMinus8

```
uint32_t _MT_ENC_CONFIG_HEVC::pixelBitDepthMinus8
```

[in]: Specifies the pixel bit depth minus 8. Should be set to 0 for 8-bit input, 2 for 10-bit input.

6.4.2.4 reserved1

```
uint32_t _MT_ENC_CONFIG_HEVC::reserved1[252]
```

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

6.4.2.5 reserved2

```
void* _MT_ENC_CONFIG_HEVC::reserved2[64]
```

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

6.4.2.6 reservedP1

```
uint32_t _MT_ENC_CONFIG_HEVC::reservedP1
```

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

6.4.2.7 reservedP2

```
uint32_t _MT_ENC_CONFIG_HEVC::reservedP2
```

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

6.4.2.8 tier

```
uint32_t _MT_ENC_CONFIG_HEVC::tier
```

[in]: Specifies the level tier of the encoded bitstream.

The documentation for this struct was generated from the following file:

- mtEncodeAPI.h

6.5 _MT_ENC_CREATE_ENCODER_PARAMS Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- uint32_t [version](#)
- [MT_ENC_DEVICE_TYPE](#) [deviceType](#)
- void * [device](#)
- uint32_t [reserved1](#) [254]
- void * [reserved2](#) [63]

6.5.1 Detailed Description

Encoder creation parameters.

6.5.2 Member Data Documentation

6.5.2.1 device

```
void* _MT_ENC_CREATE_ENCODER_PARAMS::device
```

[in]: A pointer to the client device.

6.5.2.2 deviceType

```
MT\_ENC\_DEVICE\_TYPE _MT_ENC_CREATE_ENCODER_PARAMS::deviceType
```

[in]: The device type.

6.5.2.3 reserved1

```
uint32_t _MT_ENC_CREATE_ENCODER_PARAMS::reserved1[254]
```

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

6.5.2.4 reserved2

```
void* _MT_ENC_CREATE_ENCODER_PARAMS::reserved2[63]
```

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

6.5.2.5 version

```
uint32_t _MT_ENC_CREATE_ENCODER_PARAMS::version
```

[in]: The struct version. Must be set to [MTENCAP_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.6 _MT_ENC_CREATE_OUTPUT_BUFFER Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- `uint32_t` [version](#)
- `uint32_t` [reserved](#)
- [MT_ENC_OUTPUT_PTR](#) `outputBuffer`
- `uint32_t` [reserved1](#) [62]
- `void *` [reserved2](#) [63]

6.6.1 Detailed Description

Creation parameters for output bitstream buffers.

6.6.2 Member Data Documentation

6.6.2.1 outputBuffer

```
MT_ENC_OUTPUT_PTR _MT_ENC_CREATE_OUTPUT_BUFFER::outputBuffer
```

[out]: A pointer to the output buffer.

6.6.2.2 reserved

```
uint32_t _MT_ENC_CREATE_OUTPUT_BUFFER::reserved
```

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

6.6.2.3 reserved1

```
uint32_t _MT_ENC_CREATE_OUTPUT_BUFFER::reserved1[62]
```

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

6.6.2.4 reserved2

```
void* _MT_ENC_CREATE_OUTPUT_BUFFER::reserved2[63]
```

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

6.6.2.5 version

```
uint32_t _MT_ENC_CREATE_OUTPUT_BUFFER::version
```

[in]: The struct version. Must be set to [MTENC_API_VERSION](#).

The documentation for this struct was generated from the following file:

- mtEncodeAPI.h

6.7 _MT_ENC_INIT_PARAMS Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- `uint32_t version`
- `MT_ENC_CODEC_ID encodeID`
- `MT_ENC_PRESET_ID presetID`
- `uint32_t encodeWidth`
- `uint32_t encodeHeight`
- `uint32_t frameRateNum`
- `uint32_t frameRateDen`
- `uint32_t enableEncodeAsync`
- `MT_ENC_CONFIG * encodeConfig`
- `uint32_t maxEncodeWidth`
- `uint32_t maxEncodeHeight`
- `uint32_t reserved1 [246]`
- `void * reserved2 [63]`

6.7.1 Detailed Description

Encoder initialization parameters.

6.7.2 Member Data Documentation

6.7.2.1 enableEncodeAsync

`uint32_t _MT_ENC_INIT_PARAMS::enableEncodeAsync`

[in]: Must be set to 0.

6.7.2.2 encodeConfig

`MT_ENC_CONFIG* _MT_ENC_INIT_PARAMS::encodeConfig`

[in]: Specifies the advanced codec-specific structure.

If the client sends a valid codec configuration structure, it overrides the parameters set by `MT_ENC_INIT_PARAMS::presetID`. If set to NULL, the MTEncodeAPI interface uses the `MT_ENC_INIT_PARAMS::presetID` to set the codec-specific parameters.

The client can also optionally query the MTEncodeAPI interface to get codec-specific parameters for a preset ID using `MTEncGetPresetConfig()`. It can then modify (if required) some of the codec configuration parameters and send down a custom configuration structure as part of `_MT_ENC_INIT_PARAMS`.

Even in this case the client is recommended to pass the same preset ID it has used in `MTEncGetPresetConfig()` to query the configuration structure as `MT_ENC_INIT_PARAMS::presetID`. This does not override the custom configuration structure but is used to determine other encoder HW-specific parameters not exposed in the API.

6.7.2.3 encodeHeight

`uint32_t _MT_ENC_INIT_PARAMS::encodeHeight`

[in]: Specifies the encode height. If not set, `MTEncInitEncoder()` fails.

6.7.2.4 encodeID

`MT_ENC_CODEC_ID _MT_ENC_INIT_PARAMS::encodeID`

[in]: Specifies the encode ID for which the encoder is being created. `MTEnclInitEncoder()` fails if this is not set or set to an unsupported value.

6.7.2.5 encodeWidth

`uint32_t _MT_ENC_INIT_PARAMS::encodeWidth`

[in]: Specifies the encode width. If not set, `MTEnclInitEncoder()` fails.

6.7.2.6 frameRateDen

`uint32_t _MT_ENC_INIT_PARAMS::frameRateDen`

[in]: Specifies the denominator for the frame rate used for encoding in frames per second. Frame rate = $\text{frameRateNum} / \text{frameRateDen}$.

6.7.2.7 frameRateNum

`uint32_t _MT_ENC_INIT_PARAMS::frameRateNum`

[in]: Specifies the numerator for the frame rate used for encoding in frames per second (FPS). Frame rate = $\text{frameRateNum} / \text{frameRateDen}$.

6.7.2.8 maxEncodeHeight

`uint32_t _MT_ENC_INIT_PARAMS::maxEncodeHeight`

[in]: The maximum encode height to be allowed for current encoder. The client should allocate output buffers according to this dimension for dynamic resolution changes. If set to 0, the encoder does not allow dynamic resolution changes.

6.7.2.9 maxEncodeWidth

`uint32_t _MT_ENC_INIT_PARAMS::maxEncodeWidth`

[in]: The maximum encode width to be allowed for the current encoder. The client should allocate output buffers according to this dimension for dynamic resolution changes. If set to 0, the encoder does not allow dynamic resolution changes.

6.7.2.10 presetID

`MT_ENC_PRESET_ID _MT_ENC_INIT_PARAMS::presetID`

[in]: Specifies the preset ID for encoding. If the preset ID is set, the preset configuration is applied before any other parameters.

6.7.2.11 reserved1

```
uint32_t _MT_ENC_INIT_PARAMS::reserved1[246]
```

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

6.7.2.12 reserved2

```
void* _MT_ENC_INIT_PARAMS::reserved2[63]
```

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

6.7.2.13 version

```
uint32_t _MT_ENC_INIT_PARAMS::version
```

[in]: The struct version. Must be set to [MTENCAPI_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.8 _MT_ENC_LOCK_BUFFER Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- `uint32_t` [version](#)
- `uint32_t` [frameIdx](#)
- `MT_ENC_PIC_TYPE` [pictureType](#)
- `uint32_t` [outputBufferSizeInBytes](#)
- `void *` [outputBufferPtr](#)
- `void *` [lockedOutputBuffer](#)
- `uint32_t` [reserved1](#) [252]
- `void *` [reserved2](#) [62]

6.8.1 Detailed Description

Bitstream buffer lock parameters.

6.8.2 Member Data Documentation

6.8.2.1 frameIdx

```
uint32_t _MT_ENC_LOCK_BUFFER::frameIdx
```

[out]: The frame index for which the buffer is being retrieved.

6.8.2.2 lockedOutputBuffer

```
void* _MT_ENC_LOCK_BUFFER::lockedOutputBuffer
```

[out]: A pointer to the output buffer being locked.

6.8.2.3 outputBufferPtr

```
void* _MT_ENC_LOCK_BUFFER::outputBufferPtr
```

[out]: A pointer to the generated output buffer.

6.8.2.4 outputBufferSizeInBytes

```
uint32_t _MT_ENC_LOCK_BUFFER::outputBufferSizeInBytes
```

[out]: The actual number of bytes generated and copied to the memory pointed by `outputBufferPtr`.

6.8.2.5 pictureType

```
MT_ENC_PIC_TYPE _MT_ENC_LOCK_BUFFER::pictureType
```

[out]: The picture type of the encoded picture.

6.8.2.6 reserved1

```
uint32_t _MT_ENC_LOCK_BUFFER::reserved1[252]
```

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

6.8.2.7 reserved2

```
void* _MT_ENC_LOCK_BUFFER::reserved2[62]
```

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

6.8.2.8 version

```
uint32_t _MT_ENC_LOCK_BUFFER::version
```

[in]: The struct version. Must be set to [MTENCAPI_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.9 _MT_ENC_MAP_RESOURCE Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- [uint32_t version](#)
- [MT_ENC_INPUT_RESOURCE_TYPE resourceType](#)
- [MT_ENC_INPUT_PTR resourceToMap](#)
- [MT_ENC_INPUT_PTR mappedResource](#)
- [uint32_t reserved1](#) [254]
- [void * reserved2](#) [62]

6.9.1 Detailed Description

Maps an input resource to the MT encoder input buffer.

6.9.2 Member Data Documentation

6.9.2.1 mappedResource

```
MT\_ENC\_INPUT\_PTR _MT_ENC_MAP_RESOURCE::mappedResource
```

[out]: Mapped pointer corresponding to [resourceToMap](#). This pointer must be used in [MT_ENC_PIC_PARAMS::inputBuffer](#) in [MTEncEncodeFrame\(\)](#).

6.9.2.2 reserved1

```
uint32_t _MT_ENC_MAP_RESOURCE::reserved1[254]
```

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

6.9.2.3 reserved2

```
void* _MT_ENC_MAP_RESOURCE::reserved2[62]
```

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

6.9.2.4 resourceToMap

```
MT_ENC_INPUT_PTR _MT_ENC_MAP_RESOURCE::resourceToMap
```

[in]: Handle to the resource that is being mapped.

6.9.2.5 resourceType

```
MT_ENC_INPUT_RESOURCE_TYPE _MT_ENC_MAP_RESOURCE::resourceType
```

[in]: The type of the resource to be mapped.

6.9.2.6 version

```
uint32_t _MT_ENC_MAP_RESOURCE::version
```

[in]: The struct version. Must be set to [MTENCAPI_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.10 _MT_ENC_PIC_PARAMS Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- uint32_t [version](#)
- uint32_t [inputWidth](#)
- uint32_t [inputHeight](#)
- uint32_t [inputPitch](#)
- uint32_t [encodePicFlags](#)
- [MT_ENC_BUFFER_FORMAT](#) [bufferFmt](#)
- [MT_ENC_INPUT_PTR](#) [inputBuffer](#)
- [MT_ENC_OUTPUT_PTR](#) [outputBuffer](#)
- void * [completionEvent](#)
- uint32_t [reserved0](#)
- uint32_t [roiNumber](#)
- [MT_ENCODER_ROI](#) * [roi](#)
- uint32_t [reserved1](#) [248]
- void * [reserved2](#) [60]

6.10.1 Detailed Description

Picture encoding parameters, sent on a frame-by-frame basis.

6.10.2 Member Data Documentation

6.10.2.1 bufferFmt

[MT_ENC_BUFFER_FORMAT](#) `_MT_ENC_PIC_PARAMS::bufferFmt`

[in]: Specifies the input buffer format.

6.10.2.2 completionEvent

`void* _MT_ENC_PIC_PARAMS::completionEvent`

[in]: Specifies an event to be signalled on the completion of encoding this Frame when operating in the asynchronous mode. Each output buffer should be associated with a distinct event pointer.

6.10.2.3 encodePicFlags

`uint32_t _MT_ENC_PIC_PARAMS::encodePicFlags`

[in]: Specifies bit-wise OR`ed encode pic flags. See [MT_ENC_PIC_FLAGS](#).

6.10.2.4 inputBuffer

[MT_ENC_INPUT_PTR](#) `_MT_ENC_PIC_PARAMS::inputBuffer`

[in]: Specifies an input buffer pointer. The client must use the pointer obtained from [MTEncMapResource\(\)](#).

6.10.2.5 inputHeight

`uint32_t _MT_ENC_PIC_PARAMS::inputHeight`

[in]: Specifies the input buffer height.

6.10.2.6 inputPitch

`uint32_t _MT_ENC_PIC_PARAMS::inputPitch`

[in]: Specifies the input buffer pitch. If the pitch value is unknown, set this to `inputWidth`.

6.10.2.7 inputWidth

```
uint32_t _MT_ENC_PIC_PARAMS::inputWidth
```

[in]: Specifies the input buffer width.

6.10.2.8 outputBuffer

```
MT_ENC_OUTPUT_PTR _MT_ENC_PIC_PARAMS::outputBuffer
```

[in]: Specifies a pointer to the output buffer. The client should use the pointer obtained from [MTEncCreateOutputBuffer\(\)](#).

6.10.2.9 reserved0

```
uint32_t _MT_ENC_PIC_PARAMS::reserved0
```

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

6.10.2.10 reserved1

```
uint32_t _MT_ENC_PIC_PARAMS::reserved1[248]
```

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

6.10.2.11 reserved2

```
void* _MT_ENC_PIC_PARAMS::reserved2[60]
```

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

6.10.2.12 roi

```
MT_ENCODER_ROI* _MT_ENC_PIC_PARAMS::roi
```

[in]: Specifies the input ROI array.

6.10.2.13 roiNumber

```
uint32_t _MT_ENC_PIC_PARAMS::roiNumber
```

[in]: Specifies the input ROI array number.

6.10.2.14 version

```
uint32_t _MT_ENC_PIC_PARAMS::version
```

[in]: The struct version. Must be set to [MTENCAPI_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.11 _MT_ENC_PRESET_CONFIG Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- `uint32_t` [version](#)
- `uint32_t` [reserved](#)
- `MT_ENC_CONFIG` [presetCfg](#)
- `uint32_t` [reserved1](#) [254]
- `void *` [reserved2](#) [64]

6.11.1 Detailed Description

The encoder preset configuration structure.

6.11.2 Member Data Documentation

6.11.2.1 presetCfg

```
MT\_ENC\_CONFIG _MT_ENC_PRESET_CONFIG::presetCfg
```

[out]: A preset configuration structure returned by the MTEncodeAPI interface.

6.11.2.2 reserved

```
uint32_t _MT_ENC_PRESET_CONFIG::reserved
```

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

6.11.2.3 reserved1

```
uint32_t _MT_ENC_PRESET_CONFIG::reserved1[254]
```

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

6.11.2.4 reserved2

```
void* _MT_ENC_PRESET_CONFIG::reserved2[64]
```

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

6.11.2.5 version

```
uint32_t _MT_ENC_PRESET_CONFIG::version
```

[in]: The struct version. Must be set to [MTENCAPI_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.12 _MT_ENC_QP Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- [uint32_t qplInterP](#)
- [uint32_t qplInterB](#)
- [uint32_t qplIntra](#)

6.12.1 Detailed Description

QP values for frames.

6.12.2 Member Data Documentation

6.12.2.1 qplInterB

```
uint32_t _MT_ENC_QP::qpInterB
```

[in]: The QP value for B-frame encoding.

6.12.2.2 qplInterP

```
uint32_t _MT_ENC_QP::qpInterP
```

[in]: The quantization parameter (QP) value for P-frame encoding.

6.12.2.3 qplIntra

```
uint32_t _MT_ENC_QP::qpIntra
```

[in]: The QP value for I-frame encoding.

The documentation for this struct was generated from the following file:

- `mtEncodeAPI.h`

6.13 _MT_ENC_RC_PARAMS Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- [MT_ENC_PARAMS_RC_MODE](#) `rateControlMode`
- [MT_ENC_QP](#) `constQP`
- [uint32_t](#) `averageBitRate`
- [uint32_t](#) `maxBitRate`
- [uint32_t](#) `reserved1` [56]
- `void *` [reserved2](#) [64]

6.13.1 Detailed Description

Rate control configuration parameters.

6.13.2 Member Data Documentation

6.13.2.1 averageBitRate

```
uint32_t _MT_ENC_RC_PARAMS::averageBitRate
```

[in]: Specifies the average bit rate in bits per second (bps) used for encoding.

6.13.2.2 constQP

[MT_ENC_QP](#) `_MT_ENC_RC_PARAMS::constQP`

[in]: Specifies the initial QP to be used for encoding. The values are used for all frames if in the constant QP mode.

6.13.2.3 maxBitRate

`uint32_t _MT_ENC_RC_PARAMS::maxBitRate`

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

6.13.2.4 rateControlMode

[MT_ENC_PARAMS_RC_MODE](#) `_MT_ENC_RC_PARAMS::rateControlMode`

[in]: Specifies the rate control mode. Check support for various rate control modes using [MT_ENC_PARAMS_RC_MODE](#) caps.

6.13.2.5 reserved1

`uint32_t _MT_ENC_RC_PARAMS::reserved1[56]`

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

6.13.2.6 reserved2

`void* _MT_ENC_RC_PARAMS::reserved2[64]`

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

The documentation for this struct was generated from the following file:

- `mtEncodeAPI.h`

6.14 _MT_ENCODE_API_FUNCTION_LIST Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- uint32_t [version](#)
- uint32_t [sdkVersion](#)
- PMTENC_CREATE_ENCODER [mtEncCreateEncoder](#)
- PMTENC_GET_PRESET_CONFIG [mtEncGetPresetConfig](#)
- PMTENC_INIT_ENCODER [mtEncInitEncoder](#)
- PMTENC_CREATE_OUTPUT_BUFFER [mtEncCreateOutputBuffer](#)
- PMTENC_RELEASE_OUTPUT_BUFFER [mtEncReleaseOutputBuffer](#)
- PMTENC_ENCODE_FRAME [mtEncEncodeFrame](#)
- PMTENC_LOCK_OUTPUT_BUFFER [mtEncLockOutputBuffer](#)
- PMTENC_UNLOCK_OUTPUT_BUFFER [mtEncUnlockOutputBuffer](#)
- PMTENC_MAP_RESOURCE [mtEncMapResource](#)
- PMTENC_UNMAP_RESOURCE [mtEncUnmapResource](#)
- PMTENC_RELEASE_ENCODER [mtEncReleaseEncoder](#)
- void * [reserved1](#) [245]

6.14.1 Detailed Description

A list of pointers to the MTEncodeAPI functions.

6.14.2 Member Data Documentation

6.14.2.1 mtEncCreateEncoder

PMTENC_CREATE_ENCODER _MT_ENCODE_API_FUNCTION_LIST::mtEncCreateEncoder

[out]: The client should access [MTEncCreateEncoder\(\)](#) by using this pointer.

6.14.2.2 mtEncCreateOutputBuffer

PMTENC_CREATE_OUTPUT_BUFFER _MT_ENCODE_API_FUNCTION_LIST::mtEncCreateOutputBuffer

[out]: The client should access [MTEncCreateOutputBuffer\(\)](#) by using this pointer.

6.14.2.3 mtEncEncodeFrame

PMTENC_ENCODE_FRAME _MT_ENCODE_API_FUNCTION_LIST::mtEncEncodeFrame

[out]: The client should access [MTEncEncodeFrame\(\)](#) by using this pointer.

6.14.2.4 mtEncGetPresetConfig

PMTENCGETPRESETCONFIG _MT_ENCODE_API_FUNCTION_LIST::mtEncGetPresetConfig

[out]: The client should access [MTEncGetPresetConfig\(\)](#) by using this pointer.

6.14.2.5 mtEncInitEncoder

PMTENCINITENCODER _MT_ENCODE_API_FUNCTION_LIST::mtEncInitEncoder

[out]: The client should access [MTEncInitEncoder\(\)](#) by using this pointer.

6.14.2.6 mtEncLockOutputBuffer

PMTENCLOCKOUTPUTBUFFER _MT_ENCODE_API_FUNCTION_LIST::mtEncLockOutputBuffer

[out]: The client should access [MTEncLockOutputBuffer\(\)](#) by using this pointer.

6.14.2.7 mtEncMapResource

PMTENCMAPRESOURCE _MT_ENCODE_API_FUNCTION_LIST::mtEncMapResource

[out]: The client should access [MTEncMapResource\(\)](#) by using this pointer.

6.14.2.8 mtEncReleaseEncoder

PMTENCRELEASEENCODER _MT_ENCODE_API_FUNCTION_LIST::mtEncReleaseEncoder

[out]: The client should access [MTEncReleaseEncoder\(\)](#) by using this pointer.

6.14.2.9 mtEncReleaseOutputBuffer

PMTENCRELEASEOUTPUTBUFFER _MT_ENCODE_API_FUNCTION_LIST::mtEncReleaseOutputBuffer

[out]: The client should access [MTEncReleaseOutputBuffer\(\)](#) by using this pointer.

6.14.2.10 mtEncUnlockOutputBuffer

PMTENCUNLOCKOUTPUTBUFFER _MT_ENCODE_API_FUNCTION_LIST::mtEncUnlockOutputBuffer

[out]: The client should access [MTEncUnlockOutputBuffer\(\)](#) by using this pointer.

6.14.2.11 mtEncUnmapResource

```
PMTENCUNMAPRESOURCE _MT_ENCODE_API_FUNCTION_LIST::mtEncUnmapResource
```

[out]: The client should access [MTEncUnmapResource\(\)](#) by using this pointer.

6.14.2.12 reserved1

```
void* _MT_ENCODE_API_FUNCTION_LIST::reserved1[245]
```

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

6.14.2.13 sdkVersion

```
uint32_t _MT_ENCODE_API_FUNCTION_LIST::sdkVersion
```

[out]: The SDK version.

6.14.2.14 version

```
uint32_t _MT_ENCODE_API_FUNCTION_LIST::version
```

[in]: The client should pass [MTENCAPI_VERSION](#).

The documentation for this struct was generated from the following file:

- [mtEncodeAPI.h](#)

6.15 _MT_ENCODER_ROI Struct Reference

```
#include <mtEncodeAPI.h>
```

Public Attributes

- [MT_RECTANGLE roi_rectangle](#)
Defines the ROI boundary in pixels.
- [int8_t roi_value](#)
The ROI value.

6.15.1 Detailed Description

Encoding region-of-interest (ROI).

The ROI set through this structure is applicable only to the current frame or field, so it must be sent every frame or field to be applied. The encoder will use the ROI information to adjust the QP values of the MB's that fall within the ROIs.

6.15.2 Member Data Documentation

6.15.2.1 roi_rectangle

`MT_RECTANGLE _MT_ENCODER_ROI::roi_rectangle`

Defines the ROI boundary in pixels.

The driver will map it to appropriate codec coding units. It is relative to frame coordinates for the frame case and to field coordinates for the field case.

6.15.2.2 roi_value

`int8_t _MT_ENCODER_ROI::roi_value`

The ROI value.

`roi_value` specifies ROI delta QP or ROI priority.

- The ROI delta QP is the value that will be added on top of the frame level QP.
- ROI priority specifies the priority of a region. It can be positive (more important) or negative (less important) values and is compared with a non-ROI region (taken as value 0). For example, ROI region with `roi_value` -3 is less important than the non-ROI region (`roi_value` implied to be 0) which is less important than ROI region with `roi_value` +2. For overlapping regions, `roi_value` that is first in the ROI array will have priority.

`roi_value` always specifies the ROI delta QP when
`_MT_ENC_RC_PARAMS::rateControlMode == MT_ENC_PARAMS_RC_CONSTQP`.

The documentation for this struct was generated from the following file:

- `mtEncodeAPI.h`

6.16 _MT_RECTANGLE Struct Reference

```
#include <mtEncodeAPI.h>
```

6.16.1 Detailed Description

A rectangle.

The documentation for this struct was generated from the following file:

- `mtEncodeAPI.h`

Chapter 7

File Documentation

7.1 mtEncodeAPI.h

```
1 /*****
2 * Copyright (c) 2020-2022 Moore Threads Technology Co., Ltd ("Moore Threads"). All rights reserved.
3 *
4 * This software ("this software and its documentation" or "the software") is
5 * protected by Copyright and the information contained herein is confidential.
6 *
7 * The software contained herein is PROPRIETARY to Moore Threads and is being provided under
8 * the terms and conditions of a form of Moore Threads software license agreement by and
9 * between Moore Threads and Licensee ("Moore Threads Software License Agreement") or
10 * electronically accepted by Licensee. Notwithstanding any terms or conditions to
11 * the contrary in the License Agreement, copy or disclosure
12 * of the software to any third party without the express
13 * written consent of Moore Threads is prohibited.
14 *
15 * NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY IN THE
16 * LICENSE AGREEMENT, MOORE THREADS MAKES NO REPRESENTATION ABOUT ANY
17 * WARRANTIES, INCLUDING BUT NOT LIMITED TO THE
18 * SUITABILITY OF THE SOFTWARE FOR ANY PURPOSE. IT IS
19 * PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND.
20 * MOORE THREADS DISCLAIMS ALL WARRANTIES WITH REGARD TO THE
21 * SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY,
22 * NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.
23 * NOTWITHSTANDING ANY TERMS OR CONDITIONS TO THE CONTRARY IN THE
24 * LICENSE AGREEMENT, IN NO EVENT SHALL MOORE THREADS BE LIABLE FOR ANY
25 * SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, OR ANY
26 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA, OR PROFITS,
27 * WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
28 * ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
29 * PERFORMANCE OF THE SOFTWARE.
30 *****/
31
32 #ifndef _MT_ENCODEAPI_H_
33 #define _MT_ENCODEAPI_H_
34
35 #include <stdlib.h>
36 #include <stdint.h>
37
38 #ifdef __cplusplus
39 extern "C" {
40 #endif
41
42 #define MTENCAPI __stdcall
43
44 #define MTENCAPI_MAJOR_VERSION 1
45 #define MTENCAPI_MINOR_VERSION 0
46 #define MTENCAPI_PATCH_VERSION 0
47
48 #define MTENCAPI_VERSION ((MTENCAPI_MAJOR_VERSION << 24) | (MTENCAPI_MINOR_VERSION << 16) |
49 MTENCAPI_PATCH_VERSION)
50 #define MTENC_INFINITE_GOPLength 0xFFFFFFFF
52 #define MT_ALIGN64(x) (((x) + 0x3f) & ~0x3f)
53
54 typedef void* MT_ENC_INPUT_PTR;
55 typedef void* MT_ENC_OUTPUT_PTR;
56
57 typedef enum _MT_ENC_CODEC_ID
58 {
59     MT_ENC_CODEC_ID_H264 = 0x1,
60     MT_ENC_CODEC_ID_HEVC = 0x2,
61 }
```

```

125 } MT_ENC_CODEC_ID;
126
130 typedef enum _MT_ENC_CODEC_PROFILE_ID
131 {
132     MT_ENC_CODEC_PROFILE_ID_AUTOSELECT = 0x1,
133     MT_ENC_CODEC_PROFILE_ID_BASELINE_H264 = 0x2,
134     MT_ENC_CODEC_PROFILE_ID_MAIN_H264 = 0x3,
135     MT_ENC_CODEC_PROFILE_ID_HIGH_H264 = 0x4,
136
137     MT_ENC_CODEC_PROFILE_ID_MAIN_HEVC = 0x10,
138     MT_ENC_CODEC_PROFILE_ID_MAIN_10_HEVC = 0x11,
139
140 } MT_ENC_CODEC_PROFILE_ID;
141
145 typedef enum _MT_ENC_PRESET_ID
146 {
147     MT_ENC_PRESET_ID_DEFAULT = 0x1,
148 } MT_ENC_PRESET_ID;
149
153 typedef enum _MT_ENC_PARAMS_RC_MODE
154 {
155     MT_ENC_PARAMS_RC_CONSTQP = 0x0,
156     MT_ENC_PARAMS_RC_CBR = 0x1,
157     MT_ENC_PARAMS_RC_VBR = 0x2,
158 } MT_ENC_PARAMS_RC_MODE;
159
163 typedef enum _MT_ENC_PIC_TYPE
164 {
165     MT_ENC_PIC_TYPE_I = 0x0,
166     MT_ENC_PIC_TYPE_P = 0x1,
167     MT_ENC_PIC_TYPE_B = 0x2,
168     MT_ENC_PIC_TYPE_IDR = 0x3,
169     MT_ENC_PIC_TYPE_UNKNOWN = 0xFF
170 } MT_ENC_PIC_TYPE;
171
175 typedef enum _MT_ENC_BUFFER_FORMAT
176 {
177     MT_ENC_BUFFER_FORMAT_UNDEFINED = 0x00000000,
178     MT_ENC_BUFFER_FORMAT_NV12 = 0x00000001,
179     MT_ENC_BUFFER_FORMAT_BGRA = 0x10000000,
180     MT_ENC_BUFFER_FORMAT_P010_LE = 0x00010000,
181 } MT_ENC_BUFFER_FORMAT;
182
186 typedef enum _MT_ENC_LEVEL
187 {
188     MT_ENC_LEVEL_AUTOSELECT = 0x0,
189
190     MT_ENC_LEVEL_H264_1 = 10,
191     MT_ENC_LEVEL_H264_11 = 11,
192     MT_ENC_LEVEL_H264_12 = 12,
193     MT_ENC_LEVEL_H264_13 = 13,
194     MT_ENC_LEVEL_H264_2 = 20,
195     MT_ENC_LEVEL_H264_21 = 21,
196     MT_ENC_LEVEL_H264_22 = 22,
197     MT_ENC_LEVEL_H264_3 = 30,
198     MT_ENC_LEVEL_H264_31 = 31,
199     MT_ENC_LEVEL_H264_32 = 32,
200     MT_ENC_LEVEL_H264_4 = 40,
201     MT_ENC_LEVEL_H264_41 = 41,
202     MT_ENC_LEVEL_H264_42 = 42,
203     MT_ENC_LEVEL_H264_5 = 50,
204     MT_ENC_LEVEL_H264_51 = 51,
205     MT_ENC_LEVEL_H264_52 = 52,
206
207     MT_ENC_LEVEL_HEVC_1 = 30,
208     MT_ENC_LEVEL_HEVC_2 = 60,
209     MT_ENC_LEVEL_HEVC_21 = 63,
210     MT_ENC_LEVEL_HEVC_3 = 90,
211     MT_ENC_LEVEL_HEVC_31 = 93,
212     MT_ENC_LEVEL_HEVC_4 = 120,
213     MT_ENC_LEVEL_HEVC_41 = 123,
214     MT_ENC_LEVEL_HEVC_5 = 150,
215     MT_ENC_LEVEL_HEVC_51 = 153,
216     MT_ENC_LEVEL_HEVC_52 = 156,
217     MT_ENC_LEVEL_HEVC_6 = 180,
218     MT_ENC_LEVEL_HEVC_61 = 183,
219     MT_ENC_LEVEL_HEVC_62 = 186,
220
221     MT_ENC_TIER_HEVC_MAIN = 0,
222     MT_ENC_TIER_HEVC_HIGH = 1,
223
224 } MT_ENC_LEVEL;
225
229 typedef enum _MTENCSTATUS
230 {
231     MT_ENC_SUCCESS,
232 }
233

```

```

239     MT_ENC_ERR_NO_ENCODE_DEVICE,
240
244     MT_ENC_ERR_UNSUPPORTED_DEVICE,
245
249     MT_ENC_ERR_INVALID_PARAM,
250
254     MT_ENC_ERR_OUT_OF_MEMORY,
255
262     MT_ENC_ERR_ENCODER_NOT_INITIALIZED,
263
267     MT_ENC_ERR_INVALID_VERSION,
268
272     MT_ENC_ERR_MAP_FAILED,
273
278     MT_ENC_ERR_RESOURCE_NOT_MAPPED,
279
293     MT_ENC_ERR_NEED_MORE_INPUT,
294
300     MT_ENC_ERR_ENCODER_BUSY,
301
305     MT_ENC_ERR_GENERIC,
306
307 } MT_ENC_STATUS;
308
312 typedef enum _MT_ENC_PIC_FLAGS
313 {
314     MT_ENC_PIC_FLAG_FORCEIDR          = 0x2,
316     MT_ENC_PIC_FLAG_EOS              = 0x8,
317 } MT_ENC_PIC_FLAGS;
318
322 typedef enum _MT_ENC_INPUT_RESOURCE_TYPE
323 {
324     MT_ENC_INPUT_RESOURCE_TYPE_DIRECTX = 0x0,
325 } MT_ENC_INPUT_RESOURCE_TYPE;
326
330 typedef enum _MT_ENC_DEVICE_TYPE
331 {
332     MT_ENC_DEVICE_TYPE_DIRECTX        = 0x0,
333 } MT_ENC_DEVICE_TYPE;
334
338 typedef struct _MT_ENC_CREATE_OUTPUT_BUFFER
339 {
340     uint32_t      version;
341     uint32_t      reserved;
342     MT_ENC_OUTPUT_PTR  outputBuffer;
343     uint32_t      reserved1[62];
344     void*         reserved2[63];
345 } MT_ENC_CREATE_OUTPUT_BUFFER;
346
350 typedef struct _MT_ENC_QP
351 {
352     uint32_t      qpInterP;
353     uint32_t      qpInterB;
354     uint32_t      qpIntra;
355 } MT_ENC_QP;
356
360 typedef struct _MT_ENC_RC_PARAMS
361 {
362     uint32_t      version;
363     MT_ENC_PARAMS_RC_MODE  rateControlMode;
364     MT_ENC_QP      constQP;
365     uint32_t      averageBitRate;
366     uint32_t      maxBitRate;
367     uint32_t      reserved1[56];
368     void*         reserved2[64];
369 } MT_ENC_RC_PARAMS;
370
375 typedef struct _MT_ENC_CONFIG_H264
376 {
377     uint32_t level;
378     uint32_t idrPeriod;
379     uint32_t reserved1[254];
380     void*     reserved2[64];
381 } MT_ENC_CONFIG_H264;
382
386 typedef struct _MT_ENC_CONFIG_HEVC
387 {
388     uint32_t level;
389     uint32_t tier;
390     uint32_t idrPeriod;
391     uint32_t reservedP1          : 11;
392     uint32_t pixelBitDepthMinus8 : 3;
393     uint32_t reservedP2          : 18;
394     uint32_t reserved1[252];
395     void*     reserved2[64];
396 } MT_ENC_CONFIG_HEVC;
397

```

```

401 typedef struct _MT_ENC_CODEC_CONFIG
402 {
403     MT_ENC_CONFIG_H264          h264Config;
404     MT_ENC_CONFIG_HEVC          hevcConfig;
405     uint32_t                    reserved1[1280];
406     void*                       reserved2[320];
407 } MT_ENC_CODEC_CONFIG;
408
412 typedef struct _MT_ENC_CONFIG
413 {
414     uint32_t                    version;
415     MT_ENC_CODEC_PROFILE_ID     profileID;
416     uint32_t                    gopLength;
417     int32_t                     frameIntervalP;
418     MT_ENC_RC_PARAMS            rcParams;
419     MT_ENC_CODEC_CONFIG         encodeCodecConfig;
420     uint32_t                    bDepth;
421     uint32_t                    refs;
422     uint32_t                    reserved1[250];
423     void*                       reserved2[64];
424 } MT_ENC_CONFIG;
425
429 typedef struct _MT_ENC_INIT_PARAMS
430 {
431     uint32_t                    version;
432     MT_ENC_CODEC_ID             encodeID;
433     MT_ENC_PRESET_ID            presetID;
434     uint32_t                    encodeWidth;
435     uint32_t                    encodeHeight;
436     uint32_t                    frameRateNum;
437     uint32_t                    frameRateDen;
438     uint32_t                    enableEncodeAsync;
439     MT_ENC_CONFIG*              encodeConfig;
440     uint32_t                    maxEncodeWidth;
441     uint32_t                    maxEncodeHeight;
442     uint32_t                    reserved1[246];
443     void*                       reserved2[63];
444 } MT_ENC_INIT_PARAMS;
445
450
454 typedef struct _MT_ENC_PRESET_CONFIG
455 {
456     uint32_t                    version;
457     uint32_t                    reserved;
458     MT_ENC_CONFIG presetCfg;
459     uint32_t                    reserved1[254];
460     void*                       reserved2[64];
461 } MT_ENC_PRESET_CONFIG;
462
466 typedef struct _MT_RECTANGLE
467 {
468     int16_t x;
469     int16_t y;
470     uint16_t width;
471     uint16_t height;
472 } MT_RECTANGLE;
473
481 typedef struct _MT_ENCODER_ROI
482 {
483     MT_RECTANGLE roi_rectangle;
484     int8_t roi_value;
485 } MT_ENCODER_ROI;
486
506
511 typedef struct _MT_ENC_PIC_PARAMS
512 {
513     uint32_t                    version;
514     uint32_t                    inputWidth;
515     uint32_t                    inputHeight;
516     uint32_t                    inputPitch;
517     uint32_t                    encodePicFlags;
518     MT_ENC_BUFFER_FORMAT        bufferFmt;
519     MT_ENC_INPUT_PTR            inputBuffer;
520     MT_ENC_OUTPUT_PTR           outputBuffer;
521     void*                       completionEvent;
522     uint32_t                    reserved0;
523     uint32_t                    roiNumber;
524     MT_ENCODER_ROI*             roi;
525     uint32_t                    reserved1[248];
526     void*                       reserved2[60];
527 } MT_ENC_PIC_PARAMS;
528
530
534 typedef struct _MT_ENC_LOCK_BUFFER
535 {
536     uint32_t                    version;
537     uint32_t                    frameIdx;
538     MT_ENC_PIC_TYPE             pictureType;
539     uint32_t                    outputBufferSizeInBytes;
540     void*                       outputBufferPtr;

```

```

541     void*                lockedOutputBuffer;
542     uint32_t             reserved1[252];
543     void*                reserved2[62];
544 } MT_ENC_LOCK_BUFFER;
545
546 typedef struct _MT_ENC_MAP_RESOURCE
547 {
548     uint32_t             version;
549     MT_ENC_INPUT_RESOURCE_TYPE resourceType;
550     MT_ENC_INPUT_PTR     resourceToMap;
551     MT_ENC_INPUT_PTR     mappedResource;
552     uint32_t             reserved1[254];
553     void*                reserved2[62];
554 } MT_ENC_MAP_RESOURCE;
555
556 typedef struct _MT_ENC_CREATE_ENCODER_PARAMS
557 {
558     uint32_t             version;
559     MT_ENC_DEVICE_TYPE   deviceType;
560     void*                device;
561     uint32_t             reserved1[254];
562     void*                reserved2[63];
563 } MT_ENC_CREATE_ENCODER_PARAMS;
564
565 /* END ENCODER_STRUCTURE */
566
567 // MTEncCreateEncoder
568 MTENCSTATUS MTENCAPI MTEncCreateEncoder (MT_ENC_CREATE_ENCODER_PARAMS
569     *createEncoderParams, void** encoder);
570
571 // MTEncGetPresetConfig
572 MTENCSTATUS MTENCAPI MTEncGetPresetConfig (void* encoder, MT_ENC_CODEC_ID encodeID,
573     MT_ENC_PRESET_ID presetID, MT_ENC_PRESET_CONFIG* presetConfig);
574
575 // MTEncInitEncoder
576 MTENCSTATUS MTENCAPI MTEncInitEncoder (void* encoder, MT_ENC_INIT_PARAMS*
577     initParams);
578
579 // MTEncCreateOutputBuffer
580 MTENCSTATUS MTENCAPI MTEncCreateOutputBuffer (void* encoder,
581     MT_ENC_CREATE_OUTPUT_BUFFER* createOutputBufferParams);
582
583 // MTEncReleaseOutputBuffer
584 MTENCSTATUS MTENCAPI MTEncReleaseOutputBuffer (void* encoder, MT_ENC_OUTPUT_PTR
585     outputBuffer);
586
587 // MTEncEncodeFrame
588 MTENCSTATUS MTENCAPI MTEncEncodeFrame (void* encoder, MT_ENC_PIC_PARAMS*
589     encodePicParams);
590
591 // MTEncLockOutputBuffer
592 MTENCSTATUS MTENCAPI MTEncLockOutputBuffer (void* encoder, MT_ENC_LOCK_BUFFER*
593     lockOutputBufferParams);
594
595 // MTEncUnlockOutputBuffer
596 MTENCSTATUS MTENCAPI MTEncUnlockOutputBuffer (void* encoder, MT_ENC_OUTPUT_PTR
597     outputBuffer);
598
599 // MTEncMapResource
600 MTENCSTATUS MTENCAPI MTEncMapResource (void* encoder, MT_ENC_MAP_RESOURCE*
601     mapInputResParams);
602
603 // MTEncUnmapResource
604 MTENCSTATUS MTENCAPI MTEncUnmapResource (void* encoder, MT_ENC_INPUT_PTR
605     mappedInputBuffer);
606
607 // MTEncReleaseEncoder
608 MTENCSTATUS MTENCAPI MTEncReleaseEncoder (void* encoder);
609
610 // MTEncodeAPIGetMaxSupportedVersion
611 MTENCSTATUS MTENCAPI MTEncodeAPIGetMaxSupportedVersion (uint32_t* version);
612
613 /*
614 * Defines API function pointers.
615 */
616 typedef MTENCSTATUS (MTENCAPI* PMTENC_CREATE_ENCODER) (MT_ENC_CREATE_ENCODER_PARAMS
617     *createEncoderParams, void** encoder);
618 typedef MTENCSTATUS (MTENCAPI* PMTENC_GET_PRESET_CONFIG) (void* encoder, MT_ENC_CODEC_ID encodeID,
619     MT_ENC_PRESET_ID presetID, MT_ENC_PRESET_CONFIG* presetConfig);
620 typedef MTENCSTATUS (MTENCAPI* PMTENC_INIT_ENCODER) (void* encoder, MT_ENC_INIT_PARAMS*
621     initParams);
622 typedef MTENCSTATUS (MTENCAPI* PMTENC_CREATE_OUTPUT_BUFFER) (void* encoder, MT_ENC_CREATE_OUTPUT_BUFFER*
623     createOutputBufferParams);
624 typedef MTENCSTATUS (MTENCAPI* PMTENC_RELEASE_OUTPUT_BUFFER) (void* encoder, MT_ENC_OUTPUT_PTR
625     outputBuffer);
626 typedef MTENCSTATUS (MTENCAPI* PMTENC_ENCODE_FRAME) (void* encoder, MT_ENC_PIC_PARAMS*
627     encodePicParams);

```

```

953 typedef MTENCSTATUS (MTENCAPI* PMTENCLOCKOUTPUTBUFFER)      (void* encoder, MT_ENC_LOCK_BUFFER*
    lockOutputBufferParams);
954 typedef MTENCSTATUS (MTENCAPI* PMTENCUNLOCKOUTPUTBUFFER)    (void* encoder, MT_ENC_OUTPUT_PTR
    outputBuffer);
955 typedef MTENCSTATUS (MTENCAPI* PMTENCMAPRESOURCE)            (void* encoder, MT_ENC_MAP_RESOURCE*
    mapInputResParams);
956 typedef MTENCSTATUS (MTENCAPI* PMTENCUNMAPRESOURCE)          (void* encoder, MT_ENC_INPUT_PTR
    mappedInputBuffer);
957 typedef MTENCSTATUS (MTENCAPI* PMTENCRELEASEENCODER)         (void* encoder);
958
959
961 /* END ENCODE_FUNC */
963
968 typedef struct _MT_ENCODE_API_FUNCTION_LIST
969 {
970     uint32_t                version;
971     uint32_t                sdkVersion;
972     PMTENC_CREATEENCODER    mtEncCreateEncoder;
973     PMTENC_GETPRESETCONFIG  mtEncGetPresetConfig;
974     PMTENC_INITENCODER      mtEncInitEncoder;
975     PMTENC_CREATEOUTPUTBUFFER mtEncCreateOutputBuffer;
976     PMTENC_RELEASEOUTPUTBUFFER mtEncReleaseOutputBuffer;
977     PMTENC_ENCODEFRAME      mtEncEncodeFrame;
978     PMTENC_LOCKOUTPUTBUFFER mtEncLockOutputBuffer;
979     PMTENC_UNLOCKOUTPUTBUFFER mtEncUnlockOutputBuffer;
980     PMTENC_MAPRESOURCE      mtEncMapResource;
981     PMTENC_UNMAPRESOURCE    mtEncUnmapResource;
982     PMTENC_RELEASEENCODER    mtEncReleaseEncoder;
983     void*                   reserved1[245];
984 } MT_ENCODE_API_FUNCTION_LIST;
985
986 // MTEncodeAPICreateInstance
1002 MTENCSTATUS MTENCAPI MTEncodeAPICreateInstance(MT_ENCODE_API_FUNCTION_LIST *functionList);
1003
1004 #ifdef __cplusplus
1005 }
1006 #endif
1007
1008 #endif

```

Index

- `_MT_ENCSTATUS`
 - MTEncodeAPI data structures, 14
- `_MT_ENC_ENCODER_ROI`, 48
 - `roi_rectangle`, 49
 - `roi_value`, 49
- `_MT_ENCODE_API_FUNCTION_LIST`, 45
 - `mtEncCreateEncoder`, 46
 - `mtEncCreateOutputBuffer`, 46
 - `mtEncEncodeFrame`, 46
 - `mtEncGetPresetConfig`, 46
 - `mtEncInitEncoder`, 47
 - `mtEncLockOutputBuffer`, 47
 - `mtEncMapResource`, 47
 - `mtEncReleaseEncoder`, 47
 - `mtEncReleaseOutputBuffer`, 47
 - `mtEncUnlockOutputBuffer`, 47
 - `mtEncUnmapResource`, 47
 - `reserved1`, 48
 - `sdkVersion`, 48
 - `version`, 48
- `_MT_ENC_BUFFER_FORMAT`
 - MTEncodeAPI data structures, 11
- `_MT_ENC_CODEC_CONFIG`, 25
 - `h264Config`, 25
 - `hevcConfig`, 25
 - `reserved1`, 25
 - `reserved2`, 26
- `_MT_ENC_CODEC_ID`
 - MTEncodeAPI data structures, 12
- `_MT_ENC_CODEC_PROFILE_ID`
 - MTEncodeAPI data structures, 12
- `_MT_ENC_CONFIG`, 26
 - `bDepth`, 26
 - `encodeCodecConfig`, 26
 - `frameIntervalP`, 27
 - `gopLength`, 27
 - `profileID`, 27
 - `rcParams`, 27
 - `refs`, 27
 - `reserved1`, 27
 - `reserved2`, 27
 - `version`, 28
- `_MT_ENC_CONFIG_H264`, 28
 - `idrPeriod`, 28
 - `level`, 28
 - `reserved1`, 29
 - `reserved2`, 29
- `_MT_ENC_CONFIG_HEVC`, 29
 - `idrPeriod`, 29
 - `level`, 30
 - `pixelBitDepthMinus8`, 30
 - `reserved1`, 30
 - `reserved2`, 30
 - `reservedP1`, 30
 - `reservedP2`, 30
 - `tier`, 30
- `_MT_ENC_CREATE_ENCODER_PARAMS`, 31
 - `device`, 31
 - `deviceType`, 31
 - `reserved1`, 31
 - `reserved2`, 32
 - `version`, 32
- `_MT_ENC_CREATE_OUTPUT_BUFFER`, 32
 - `outputBuffer`, 32
 - `reserved`, 33
 - `reserved1`, 33
 - `reserved2`, 33
 - `version`, 33
- `_MT_ENC_DEVICE_TYPE`
 - MTEncodeAPI data structures, 12
- `_MT_ENC_INIT_PARAMS`, 33
 - `enableEncodeAsync`, 34
 - `encodeConfig`, 34
 - `encodeHeight`, 34
 - `encodeID`, 34
 - `encodeWidth`, 35
 - `frameRateDen`, 35
 - `frameRateNum`, 35
 - `maxEncodeHeight`, 35
 - `maxEncodeWidth`, 35
 - `presetID`, 35
 - `reserved1`, 35
 - `reserved2`, 36
 - `version`, 36
- `_MT_ENC_INPUT_RESOURCE_TYPE`
 - MTEncodeAPI data structures, 12
- `_MT_ENC_LEVEL`
 - MTEncodeAPI data structures, 12
- `_MT_ENC_LOCK_BUFFER`, 36
 - `frameIdx`, 36
 - `lockedOutputBuffer`, 37
 - `outputBufferPtr`, 37
 - `outputBufferSizeInBytes`, 37
 - `pictureType`, 37
 - `reserved1`, 37
 - `reserved2`, 37
 - `version`, 37
- `_MT_ENC_MAP_RESOURCE`, 38
 - `mappedResource`, 38
 - `reserved1`, 38
 - `reserved2`, 38
 - `resourceToMap`, 39
 - `resourceType`, 39
 - `version`, 39
- `_MT_ENC_PARAMS_RC_MODE`
 - MTEncodeAPI data structures, 13
- `_MT_ENC_PIC_FLAGS`
 - MTEncodeAPI data structures, 13

[_MT_ENC_PIC_PARAMS, 39](#)
[bufferFmt, 40](#)
[completionEvent, 40](#)
[encodePicFlags, 40](#)
[inputBuffer, 40](#)
[inputHeight, 40](#)
[inputPitch, 40](#)
[inputWidth, 40](#)
[outputBuffer, 41](#)
[reserved0, 41](#)
[reserved1, 41](#)
[reserved2, 41](#)
[roi, 41](#)
[roiNumber, 41](#)
[version, 41](#)
[_MT_ENC_PIC_TYPE](#)
[MTEncodeAPI data structures, 13](#)
[_MT_ENC_PRESET_CONFIG, 42](#)
[presetCfg, 42](#)
[reserved, 42](#)
[reserved1, 42](#)
[reserved2, 43](#)
[version, 43](#)
[_MT_ENC_PRESET_ID](#)
[MTEncodeAPI data structures, 14](#)
[_MT_ENC_QP, 43](#)
[qpInterB, 43](#)
[qpInterP, 43](#)
[qpIntra, 44](#)
[_MT_ENC_RC_PARAMS, 44](#)
[averageBitRate, 44](#)
[constQP, 44](#)
[maxBitRate, 45](#)
[rateControlMode, 45](#)
[reserved1, 45](#)
[reserved2, 45](#)
[_MT_RECTANGLE, 49](#)
[averageBitRate](#)
[_MT_ENC_RC_PARAMS, 44](#)
[bDepth](#)
[_MT_ENC_CONFIG, 26](#)
[bufferFmt](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[completionEvent](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[constQP](#)
[_MT_ENC_RC_PARAMS, 44](#)
[device](#)
[_MT_ENC_CREATE_ENCODER_PARAMS, 31](#)
[deviceType](#)
[_MT_ENC_CREATE_ENCODER_PARAMS, 31](#)
[enableEncodeAsync](#)
[_MT_ENC_INIT_PARAMS, 34](#)
[encodeCodecConfig](#)
[_MT_ENC_CONFIG, 26](#)
[encodeConfig](#)
[_MT_ENC_INIT_PARAMS, 34](#)
[encodeHeight](#)
[_MT_ENC_INIT_PARAMS, 34](#)
[encodeID](#)
[_MT_ENC_INIT_PARAMS, 34](#)
[encodePicFlags](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[encodeWidth](#)
[_MT_ENC_INIT_PARAMS, 35](#)
[frameIdx](#)
[_MT_ENC_LOCK_BUFFER, 36](#)
[frameIntervalP](#)
[_MT_ENC_CONFIG, 27](#)
[frameRateDen](#)
[_MT_ENC_INIT_PARAMS, 35](#)
[frameRateNum](#)
[_MT_ENC_INIT_PARAMS, 35](#)
[gopLength](#)
[_MT_ENC_CONFIG, 27](#)
[h264Config](#)
[_MT_ENC_CODEC_CONFIG, 25](#)
[hevcConfig](#)
[_MT_ENC_CODEC_CONFIG, 25](#)
[idrPeriod](#)
[_MT_ENC_CONFIG_H264, 28](#)
[_MT_ENC_CONFIG_HEVC, 29](#)
[inputBuffer](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[inputHeight](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[inputPitch](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[inputWidth](#)
[_MT_ENC_PIC_PARAMS, 40](#)
[level](#)
[_MT_ENC_CONFIG_H264, 28](#)
[_MT_ENC_CONFIG_HEVC, 30](#)
[lockedOutputBuffer](#)
[_MT_ENC_LOCK_BUFFER, 37](#)
[mappedResource](#)
[_MT_ENC_MAP_RESOURCE, 38](#)
[maxBitRate](#)
[_MT_ENC_RC_PARAMS, 45](#)
[maxEncodeHeight](#)
[_MT_ENC_INIT_PARAMS, 35](#)
[maxEncodeWidth](#)
[_MT_ENC_INIT_PARAMS, 35](#)
[MT_ENC_BUFFER_FORMAT](#)
[MTEncodeAPI data structures, 7](#)
[MT_ENC_BUFFER_FORMAT_BGRA](#)
[MTEncodeAPI data structures, 11](#)
[MT_ENC_BUFFER_FORMAT_NV12](#)
[MTEncodeAPI data structures, 11](#)
[MT_ENC_BUFFER_FORMAT_P010_LE](#)
[MTEncodeAPI data structures, 11](#)
[MT_ENC_BUFFER_FORMAT_UNDEFINED](#)
[MTEncodeAPI data structures, 11](#)
[MT_ENC_CODEC_CONFIG](#)
[MTEncodeAPI data structures, 7](#)
[MT_ENC_CODEC_ID](#)
[MTEncodeAPI data structures, 8](#)

MT_ENC_CODEC_ID_H264	MTEncodeAPI data structures, 12
MT_ENC_CODEC_ID_HEVC	MTEncodeAPI data structures, 12
MT_ENC_CODEC_PROFILE_ID	MTEncodeAPI data structures, 8
MT_ENC_CONFIG	MTEncodeAPI data structures, 8
MT_ENC_CONFIG_H264	MTEncodeAPI data structures, 8
MT_ENC_CONFIG_HEVC	MTEncodeAPI data structures, 8
MT_ENC_CREATE_ENCODER_PARAMS	MTEncodeAPI data structures, 8
MT_ENC_CREATE_OUTPUT_BUFFER	MTEncodeAPI data structures, 8
MT_ENC_DEVICE_TYPE	MTEncodeAPI data structures, 8
MT_ENC_DEVICE_TYPE_DIRECTX	MTEncodeAPI data structures, 12
MT_ENC_ERR_ENCODER_BUSY	MTEncodeAPI data structures, 15
MT_ENC_ERR_ENCODER_NOT_INITIALIZED	MTEncodeAPI data structures, 14
MT_ENC_ERR_GENERIC	MTEncodeAPI data structures, 15
MT_ENC_ERR_INVALID_PARAM	MTEncodeAPI data structures, 14
MT_ENC_ERR_INVALID_VERSION	MTEncodeAPI data structures, 14
MT_ENC_ERR_MAP_FAILED	MTEncodeAPI data structures, 14
MT_ENC_ERR_NEED_MORE_INPUT	MTEncodeAPI data structures, 14
MT_ENC_ERR_NO_ENCODE_DEVICE	MTEncodeAPI data structures, 14
MT_ENC_ERR_OUT_OF_MEMORY	MTEncodeAPI data structures, 14
MT_ENC_ERR_RESOURCE_NOT_MAPPED	MTEncodeAPI data structures, 14
MT_ENC_ERR_UNSUPPORTED_DEVICE	MTEncodeAPI data structures, 14
MT_ENC_INIT_PARAMS	MTEncodeAPI data structures, 9
MT_ENC_INPUT_PTR	MTEncodeAPI data structures, 9
MT_ENC_INPUT_RESOURCE_TYPE	MTEncodeAPI data structures, 9
MT_ENC_INPUT_RESOURCE_TYPE_DIRECTX	MTEncodeAPI data structures, 12
MT_ENC_LEVEL	MTEncodeAPI data structures, 9
MT_ENC_LOCK_BUFFER	MTEncodeAPI data structures, 9
MT_ENC_MAP_RESOURCE	MTEncodeAPI data structures, 9
MT_ENC_OUTPUT_PTR	MTEncodeAPI data structures, 9
MT_ENC_PARAMS_RC_CBR	MTEncodeAPI data structures, 13
MT_ENC_PARAMS_RC_CONSTQP	MTEncodeAPI data structures, 13
MT_ENC_PARAMS_RC_MODE	MTEncodeAPI data structures, 9
	MT_ENC_PARAMS_RC_VBR
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_FLAG_EOS
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_FLAG_FORCEIDR
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_FLAGS
	MTEncodeAPI data structures, 10
	MT_ENC_PIC_PARAMS
	MTEncodeAPI data structures, 10
	MT_ENC_PIC_TYPE
	MTEncodeAPI data structures, 10
	MT_ENC_PIC_TYPE_B
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_TYPE_I
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_TYPE_IDR
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_TYPE_P
	MTEncodeAPI data structures, 13
	MT_ENC_PIC_TYPE_UNKNOWN
	MTEncodeAPI data structures, 13
	MT_ENC_PRESET_CONFIG
	MTEncodeAPI data structures, 10
	MT_ENC_PRESET_ID
	MTEncodeAPI data structures, 10
	MT_ENC_PRESET_ID_DEFAULT
	MTEncodeAPI data structures, 14
	MT_ENC_QP
	MTEncodeAPI data structures, 10
	MT_ENC_RC_PARAMS
	MTEncodeAPI data structures, 10
	MT_ENC_SUCCESS
	MTEncodeAPI data structures, 14
	MT_ENCODE_API_FUNCTION_LIST
	MTEncodeAPI data structures, 10
	MT_ENCODER_ROI
	MTEncodeAPI data structures, 11
	MT_RECTANGLE
	MTEncodeAPI data structures, 11
	MTENC_INFINITE_GOPLength
	MTEncodeAPI data structures, 7
	MTENCAPI_VERSION
	MTEncodeAPI data structures, 7
	MTEncCreateEncoder
	MTEncodeAPI functions, 16
	mtEncCreateEncoder
	_MT_ENCODE_API_FUNCTION_LIST, 46
	MTEncCreateOutputBuffer
	MTEncodeAPI functions, 16
	mtEncCreateOutputBuffer
	_MT_ENCODE_API_FUNCTION_LIST, 46
	MTEncEncodeFrame
	MTEncodeAPI functions, 17
	mtEncEncodeFrame
	_MT_ENCODE_API_FUNCTION_LIST, 46
	MTEncGetPresetConfig
	MTEncodeAPI functions, 17
	mtEncGetPresetConfig
	_MT_ENCODE_API_FUNCTION_LIST, 46
	MTEnclnitEncoder
	MTEncodeAPI functions, 18

mtEncInitEncoder
 _MT_ENCODE_API_FUNCTION_LIST, 47
 MTEncLockOutputBuffer
 MTEncodeAPI functions, 19
 mtEncLockOutputBuffer
 _MT_ENCODE_API_FUNCTION_LIST, 47
 MTEncMapResource
 MTEncodeAPI functions, 20
 mtEncMapResource
 _MT_ENCODE_API_FUNCTION_LIST, 47
 MTEncodeAPI data structures, 5
 _MTENCSTATUS, 14
 _MT_ENC_BUFFER_FORMAT, 11
 _MT_ENC_CODEC_ID, 12
 _MT_ENC_CODEC_PROFILE_ID, 12
 _MT_ENC_DEVICE_TYPE, 12
 _MT_ENC_INPUT_RESOURCE_TYPE, 12
 _MT_ENC_LEVEL, 12
 _MT_ENC_PARAMS_RC_MODE, 13
 _MT_ENC_PIC_FLAGS, 13
 _MT_ENC_PIC_TYPE, 13
 _MT_ENC_PRESET_ID, 14
 MT_ENC_BUFFER_FORMAT, 7
 MT_ENC_BUFFER_FORMAT_BGRA, 11
 MT_ENC_BUFFER_FORMAT_NV12, 11
 MT_ENC_BUFFER_FORMAT_P010_LE, 11
 MT_ENC_BUFFER_FORMAT_UNDEFINED, 11
 MT_ENC_CODEC_CONFIG, 7
 MT_ENC_CODEC_ID, 8
 MT_ENC_CODEC_ID_H264, 12
 MT_ENC_CODEC_ID_HEVC, 12
 MT_ENC_CODEC_PROFILE_ID, 8
 MT_ENC_CONFIG, 8
 MT_ENC_CONFIG_H264, 8
 MT_ENC_CONFIG_HEVC, 8
 MT_ENC_CREATE_ENCODER_PARAMS, 8
 MT_ENC_CREATE_OUTPUT_BUFFER, 8
 MT_ENC_DEVICE_TYPE, 8
 MT_ENC_DEVICE_TYPE_DIRECTX, 12
 MT_ENC_ERR_ENCODER_BUSY, 15
 MT_ENC_ERR_ENCODER_NOT_INITIALIZED, 14
 MT_ENC_ERR_GENERIC, 15
 MT_ENC_ERR_INVALID_PARAM, 14
 MT_ENC_ERR_INVALID_VERSION, 14
 MT_ENC_ERR_MAP_FAILED, 14
 MT_ENC_ERR_NEED_MORE_INPUT, 14
 MT_ENC_ERR_NO_ENCODE_DEVICE, 14
 MT_ENC_ERR_OUT_OF_MEMORY, 14
 MT_ENC_ERR_RESOURCE_NOT_MAPPED, 14
 MT_ENC_ERR_UNSUPPORTED_DEVICE, 14
 MT_ENC_INIT_PARAMS, 9
 MT_ENC_INPUT_PTR, 9
 MT_ENC_INPUT_RESOURCE_TYPE, 9
 MT_ENC_INPUT_RESOURCE_TYPE_DIRECTX, 12
 MT_ENC_LEVEL, 9
 MT_ENC_LOCK_BUFFER, 9
 MT_ENC_MAP_RESOURCE, 9
 MT_ENC_OUTPUT_PTR, 9
 MT_ENC_PARAMS_RC_CBR, 13
 MT_ENC_PARAMS_RC_CONSTQP, 13
 MT_ENC_PARAMS_RC_MODE, 9
 MT_ENC_PARAMS_RC_VBR, 13
 MT_ENC_PIC_FLAG_EOS, 13
 MT_ENC_PIC_FLAG_FORCEIDR, 13
 MT_ENC_PIC_FLAGS, 10
 MT_ENC_PIC_PARAMS, 10
 MT_ENC_PIC_TYPE, 10
 MT_ENC_PIC_TYPE_B, 13
 MT_ENC_PIC_TYPE_I, 13
 MT_ENC_PIC_TYPE_IDR, 13
 MT_ENC_PIC_TYPE_P, 13
 MT_ENC_PIC_TYPE_UNKNOWN, 13
 MT_ENC_PRESET_CONFIG, 10
 MT_ENC_PRESET_ID, 10
 MT_ENC_PRESET_ID_DEFAULT, 14
 MT_ENC_QP, 10
 MT_ENC_RC_PARAMS, 10
 MT_ENC_SUCCESS, 14
 MT_ENCODE_API_FUNCTION_LIST, 10
 MT_ENCODER_ROI, 11
 MT_RECTANGLE, 11
 MTENC_INFINITE_GOPLength, 7
 MTENCAPI_VERSION, 7
 MTENCSTATUS, 11
 MTEncodeAPI functions, 15
 MTEncCreateEncoder, 16
 MTEncCreateOutputBuffer, 16
 MTEncEncodeFrame, 17
 MTEncGetPresetConfig, 17
 MTEncInitEncoder, 18
 MTEncLockOutputBuffer, 19
 MTEncMapResource, 20
 MTEncodeAPICreateInstance, 21
 MTEncodeAPIGetMaxSupportedVersion, 21
 MTEncReleaseEncoder, 22
 MTEncReleaseOutputBuffer, 22
 MTEncUnlockOutputBuffer, 23
 MTEncUnmapResource, 23
 MTEncodeAPICreateInstance
 MTEncodeAPI functions, 21
 MTEncodeAPIGetMaxSupportedVersion
 MTEncodeAPI functions, 21
 MTEncReleaseEncoder
 MTEncodeAPI functions, 22
 mtEncReleaseEncoder
 _MT_ENCODE_API_FUNCTION_LIST, 47
 MTEncReleaseOutputBuffer
 MTEncodeAPI functions, 22
 mtEncReleaseOutputBuffer
 _MT_ENCODE_API_FUNCTION_LIST, 47
 MTENCSTATUS
 MTEncodeAPI data structures, 11
 MTEncUnlockOutputBuffer
 MTEncodeAPI functions, 23
 mtEncUnlockOutputBuffer
 _MT_ENCODE_API_FUNCTION_LIST, 47
 MTEncUnmapResource
 MTEncodeAPI functions, 23
 mtEncUnmapResource
 _MT_ENCODE_API_FUNCTION_LIST, 47
 outputBuffer
 _MT_ENC_CREATE_OUTPUT_BUFFER, 32
 _MT_ENC_PIC_PARAMS, 41
 outputBufferPtr
 _MT_ENC_LOCK_BUFFER, 37
 outputBufferSizeInBytes

_MT_ENC_LOCK_BUFFER, 37	resourceType
pictureType	_MT_ENC_MAP_RESOURCE, 39
_MT_ENC_LOCK_BUFFER, 37	roi
pixelBitDepthMinus8	_MT_ENC_PIC_PARAMS, 41
_MT_ENC_CONFIG_HEVC, 30	roi_rectangle
presetCfg	_MT_ENCODER_ROI, 49
_MT_ENC_PRESET_CONFIG, 42	roi_value
presetID	_MT_ENCODER_ROI, 49
_MT_ENC_INIT_PARAMS, 35	roiNumber
profileID	_MT_ENC_PIC_PARAMS, 41
_MT_ENC_CONFIG, 27	sdkVersion
qplInterB	_MT_ENCODE_API_FUNCTION_LIST, 48
_MT_ENC_QP, 43	tier
qplInterP	_MT_ENC_CONFIG_HEVC, 30
_MT_ENC_QP, 43	version
qplIntra	_MT_ENCODE_API_FUNCTION_LIST, 48
_MT_ENC_QP, 44	_MT_ENC_CONFIG, 28
rateControlMode	_MT_ENC_CREATE_ENCODER_PARAMS, 32
_MT_ENC_RC_PARAMS, 45	_MT_ENC_CREATE_OUTPUT_BUFFER, 33
rcParams	_MT_ENC_INIT_PARAMS, 36
_MT_ENC_CONFIG, 27	_MT_ENC_LOCK_BUFFER, 37
refs	_MT_ENC_MAP_RESOURCE, 39
_MT_ENC_CONFIG, 27	_MT_ENC_PIC_PARAMS, 41
reserved	_MT_ENC_PRESET_CONFIG, 43
_MT_ENC_CREATE_OUTPUT_BUFFER, 33	
_MT_ENC_PRESET_CONFIG, 42	
reserved0	
_MT_ENC_PIC_PARAMS, 41	
reserved1	
_MT_ENCODE_API_FUNCTION_LIST, 48	
_MT_ENC_CODEC_CONFIG, 25	
_MT_ENC_CONFIG, 27	
_MT_ENC_CONFIG_H264, 29	
_MT_ENC_CONFIG_HEVC, 30	
_MT_ENC_CREATE_ENCODER_PARAMS, 31	
_MT_ENC_CREATE_OUTPUT_BUFFER, 33	
_MT_ENC_INIT_PARAMS, 35	
_MT_ENC_LOCK_BUFFER, 37	
_MT_ENC_MAP_RESOURCE, 38	
_MT_ENC_PIC_PARAMS, 41	
_MT_ENC_PRESET_CONFIG, 42	
_MT_ENC_RC_PARAMS, 45	
reserved2	
_MT_ENC_CODEC_CONFIG, 26	
_MT_ENC_CONFIG, 27	
_MT_ENC_CONFIG_H264, 29	
_MT_ENC_CONFIG_HEVC, 30	
_MT_ENC_CREATE_ENCODER_PARAMS, 32	
_MT_ENC_CREATE_OUTPUT_BUFFER, 33	
_MT_ENC_INIT_PARAMS, 36	
_MT_ENC_LOCK_BUFFER, 37	
_MT_ENC_MAP_RESOURCE, 38	
_MT_ENC_PIC_PARAMS, 41	
_MT_ENC_PRESET_CONFIG, 43	
_MT_ENC_RC_PARAMS, 45	
reservedP1	
_MT_ENC_CONFIG_HEVC, 30	
reservedP2	
_MT_ENC_CONFIG_HEVC, 30	
resourceToMap	
_MT_ENC_MAP_RESOURCE, 39	