SDK API Reference Manual

for HEVC GPU Assist APIs

API Version 1.13

LEGAL DISCLAIMER

THIS DOCUMENT CONTAINS INFORMATION ON PRODUCTS IN THE DESIGN PHASE OF DEVELOPMENT.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Table of Contents

Overview

The Intel® Media Server Studio – SDK, further referred to as the SDK, is a software development library that exposes the media acceleration capabilities of Intel platforms for decoding, encoding and video preprocessing. The API library covers a wide range of Intel platforms.

This document describes an API providing access to hardware-accelerated functions which can be used in an H265 (HEVC) encoder. Please refer to the *SDK API Reference Manual* for a complete description of the API.

Document Conventions

The Intel® Media Server Studio - SDK API uses the Verdana typeface for normal prose. With the exception of section headings and the table of contents, all code-related items appear in the `Courier New` typeface (`mxfStatus` and `MFXInit)`. All class-related items appear in all cap boldface, such as **DECODE** and **ENCODE**. Member functions appear in initial cap boldface, such as **Init** and **Reset**, and these refer to members of all classes, **DECODE**, **ENCODE** and **VPP**. Hyperlinks appear in underlined boldface, such as **`mfxStatus`**.

Acronyms and Abbreviations

FEI	Flexible Encode Infrastructure
MV	Motion Vector

Architecture

The HEVC GPU Assist APIs provide access to a set of GPU-accelerated functions which produce useful information for encoding H265 video. This functionality is implemented in the H265 Flexible Infrastructure Encoder (FEI) plugin. The encoder sends YUV frames (typically one source frame and one or more reconstructed reference frames) to the FEI plugin and specifies one or more processing steps to perform. Currently H265 FEI can produce candidate intra prediction modes, intra and inter distortion estimates, inter motion vectors, and half-pel interpolation of reference frames using a fast approximation of the H265 interpolation filter. An encoder may choose to perform additional processing in software to refine the output of FEI.

FEI runs asynchronously on the GPU which allows parallel CPU-GPU processing within a single frame. FEI also supports up to one frame of lookahead, allowing an encoder to begin GPU processing on the next frame to be encoded while the CPU completes encoding and reconstruction of the current frame.

FEI uses synchronization objects to signal when processing has completed. Although FEI can be run as a serialized pre-processing stage prior to encoding each frame, the greatest performance benefits will be realized by running FEI in parallel with CPU processing whenever possible.

Development of the FEI plugin is ongoing and new functionality will be added in future versions. The following are some important known limitations of the current version:

- Half-pel and quarter-pel motion vector refinement is performed using a fast approximation of the standard HEVC interpolation filters, so the distortion estimates may differ from an implementation which uses the standard filters.

- Motion vectors are only calculated for a subset of valid block sizes. The include file **`mfxfeih265.h`** contains the list of currently supported sizes.

- The plugin supports video resolutions up to 3840x2160.

- Intra distortion estimates are provided for 16x16 blocks. This is a rough estimate of distortion which can be calculated very quickly, and it is primarily useful for deciding whether or not intra coding should be considered for a given region. Distortion estimation employs a weighted transform so results may differ from an analysis using SAD.

- Intra angular mode decisions are calculated using a fast approximation to an exhaustive search, so FEI will not necessarily select the same "best mode" as a full search which calculates SAD for every direction. Currently only the single best mode is returned, but the API is designed to permit a sorted list of multiple candidate modes to be calculated in future versions. (`MFX_FEI_H265_MAX_INTRA_MODES`).

- Half-pel reference frames are produced with a fast approximation to the standard 8-tap HEVC interpolation filter, so the interpolated output will not be identical.

- Output buffers containing half-pel interpolated reference frames may be overwritten by subsequent FEI operations, so before initiating a new FEI operation the application should either finish use of this data or copy the output frame to a separate buffer for later CPU processing.

Programming Guide

This chapter describes the concepts used in programming the HEVC GPU Assist APIs for Intel® Media Server Studio - SDK.

The application must use the include file, **mfxfeih265.h** and **mfxvideo.h** (for C programming), or **mfxvideo++.h** (for C++ programming), and link the Intel® Media Server Studio - SDK static dispatcher library, **libmfx.lib** or **libmfx.a**. If the application is written in C then **libstdc++.a** library should also be linked.

Include these files:

```
      #include "mfxvideo.h"     /* The SDK include file */
      #include "mfxfeih265.h"   /* H265 FEI extension include file */
      #include "mfxvideo++.h"   /* Optional for C++ development */
```

Link this library:

```
      libmfx.lib                /* The SDK static dispatcher library */
```

or

```
      libmfx.a                  /* The SDK static dispatcher library */
```

The HEVC GPU Assist APIs are built upon the concept of extension buffers, and most of the configuration parameters and video data are passed in such buffers. Usually functions work with a list of such buffers at the input and output. For example, **MFXVideoENC_ProcessFrameAsync** function receives a **mfxENCInput** structure and outputs a **mfxENCOutput** structure. Both of these structures are simply lists of extension buffers, with **mfxENCInput** also holding input and reference frames. Sample code is provided to illustrate the process of loading and calling the H265 FEI plugin.

SDK API Reference Manual has more information about handling of extension buffers. In short – an extension buffer is a special SDK structure that holds an **mfxExtBuffer** value as its first member. This value holds the unique buffer ID and buffer size. The application should allocate this structure, properly set ID and size and then "attach" this buffer to one of the other structures, for example **mfxVideoParam** or **mfxENCInput**. "Attach" means to put a pointer to this extension buffer to the **ExtParam** array and to increase buffer counter **NumExtParam**. It is very important to zero all reserved fields in the extension buffers to ensure seamless future extensions.

Extension buffers may be used on any stages of the SDK pipeline – during initialization, at runtime and at reset. There are many limitations for when and how particular extension buffers may be used, please refer to the buffer description for details.

To use the HEVC GPU Assist APIs, the encoder first initializes the SDK session and loads the H265 FEI plugin. It then configures **mfxVideoParam** and passes this struct to

MFXVideoEnc_Init. The main encode loop generally loads one source frame per pass, and calls **MFXVideoENC_ProcessFrameAsync** one or more times per frame.

MFXVideoENC_ProcessFrameAsync spawns a new task for the specified set of FEI operations and then returns immediately. FEI processing takes place asynchronously on the GPU, so an updated sync object **mfxSyncPoint** is returned with each call. This will be used later in a call to **MFXVideoCORE_SyncOperation** to ensure that GPU processing is complete.

For intra-only processing (prediction mode selection and distortion estimation) only the source YUV frame is required. Both of these operations process a single frame at once, so should be called at most once per encoded frame.

For inter-processing (motion estimation and fast half-pel interpolation) source and reference frames are required. H265 supports multiple reference frames, so these operations can be called multiple times per source frame (subject to the upper limits specified in **mfxfeih265.h**) It is the caller's responsiblity to ensure that reference frames are available (i.e. reconstruction is complete) and to maintain the decoded picture buffer state.

Before using the output data from **MFXVideoENC_ProcessFrameAsync** the encoder must call **MFXVideoCORE_SyncOperation** When this function returns, the application can safely use the output data for the specified set of FEI operations, which is returned in **mfxFEIH265Output**.

The HEVC GPU Assist APIs require a compatible GPU (Intel® HD Graphics 4600 or later, Intel® Iris Pro 5200 recommended) with up-to-date graphics drivers (https://downloadcenter.intel.com). No software fallback is provided.

Function Reference

This section describes HEVC GPU Assist API functions and their operations. Refer to the *SDK API Reference Manual* for a description of other functions which are not specific to HEVC GPU Assist APIs.

In each function description, only commonly used status codes are documented. The function may return additional status codes, such as **MFX_ERR_INVALID_HANDLE** or **MFX_ERR_NULL_PTR**, in certain case. See the **mfxStatus** enumerator for a list of all status codes.

MFXVideoENC_Init

Syntax

```
mfxStatus MFXVideoENC_Init(mfxSession session, mfxVideoParam *par);
```

Parameters

session	SDK session handle
par	Pointer to the mfxVideoParam structure

Description

This function initializes H265 FEI. At minimum, the following fields in **mfxVideoParam** should be filled by the caller:

par->mfx.FrameInfo.Width	Width of video in pixels
par->mfx.FrameInfo.Height	Height of video in pixels
par->mfx.NumRefFrame	Maximum number of reference frames for ME

Video width and height should be multiples of 16 pixels (pad input frames if necessary).

The number of reference frames must be <= MFX_FEI_H265_MAX_NUM_REF_FRAMES

mfxExtFEIH265Param contains additional parameters specific to H265 FEI and should be attached to **mfxVideoParam**. Refer to the *SDK API Reference Manual* for a complete description of this function and the **mfxVideoParam** structure.

Return Status

MFX_ERR_NONE	The function completed successfully.

Change History

This function is available since SDK API 1.13.

MFXVideoENC_Reset

Syntax

```
mfxStatus MFXVideoENC_Reset(mfxSession session, mfxVideoParam *par);
```

Parameters

`session`	SDK session handle
`par`	Pointer to the mfxVideoParam structure

Description

This function resets H265 FEI. Refer to the *SDK API Reference Manual* for a complete description of this function.

Return Status

`MFX_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.13.

MFXVideoENC_Close

Syntax

```
mfxStatus MFXVideoENC_Close(mfxSession session);
```

Parameters

`session`	SDK session handle

Description

This function closes H265 FEI. Refer to the *SDK API Reference Manual* for a complete description of this function.

Return Status

`MFX_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.13.

MFXVideoENC_ProcessFrameAsync

Syntax

```
mfxStatus MFXVideoENC_ProcessFrameAsync(
                    mfxSession session, mfxENCInput *in,
                    mfxENCOutput *out, mfxSyncPoint *syncp);
```

Parameters

`session`	SDK session handle
`in`	Pointer to the input parameters
`out`	Pointer to the output parameters
`syncp`	Pointer to the sync point associated with this operation

Description

This function initiates the H265 FEI processing specified by the **`FEIOp`** field in the **`mfxExtFEIH265Input`** structure. The results will be returned in the **`mfxExtFEIH265Output`** structure. Note that this data cannot be safely accessed by the caller until AFTER calling **`MFXVideoCORE_SyncOperation`** because FEI processing occurs asynchronously. The same **`out`** pointer should be passed to every call to **`MFXVideoEnc_ProcessFrameAsync`** which operates on the same source frame, although the members of that struct may only be accessed after the relevant call to **`MFXVideoCORE_SyncOperation`** returns.

Multiple FEI operations may be specified with one call to **`MFXVideoEnc_ProcessFrameAsync`** by OR'ing operations in the **`FEIOp`** parameter (e.g. `MFX_FEI_H265_OP_INTRA_MODE | MFX_FEI_H265_OP_INTRA_DIST`). But only one forward and one backward reference are supported for motion estimation and interpolation. To perform multi-reference search the application should call this function several times.

Refer to the *SDK API Reference Manual* for a complete description of this function.

Return Status

`MFX_ERR_NONE`	The function completed successfully.

Change History

This function is available since SDK API 1.13.

MFXVideoCORE_SyncOperation

Syntax

```
mfxStatus MFXVideoCORE_SyncOperation(mfxSession session,
                      mfxSyncPoint syncp, mfxU32 wait);
```

Parameters

`session`	SDK session handle
`syncp`	Sync point
`wait`	Wait time in milliseconds

Description

This function blocks until asynchronous H265 FEI processing completes. **`syncp`** is the sync object returned from a call to **`MFXVideoEnc_ProcessFrameAsync`**. This function will return when all processing associated with this sync point has completed, or when the maximum wait time has elapsed. If `MFX_ERR_NONE` is returned then the relevant results of H265 FEI processing can safely be accessed by the application.

Refer to the *SDK API Reference Manual* for a complete description of this function.

Return Status

`MFX_ERR_NONE`	The function completed successfully.
`MFX_WRN_IN_EXECUTION`	The wait time expired before the operation completed. **`MFXVideoCORE_SyncOperation`** must be called again before accessing the output.

Change History

This function is available since SDK API 1.13.

Structure Reference

In the following structure references, all reserved fields must be zero.

mfxExtFEIH265Param

Definition

```
typedef struct
{
    mfxExtBuffer       Header;

    mfxU32 MaxCUSize;
    mfxU32 MPMode;
    mfxU32 NumIntraModes;

    mfxU16             reserved[22];
} mfxExtFEIH265Param;
```

Description

This buffer contains parameters to configure the H265 FEI plugin.

Members

`Header.BufferId`	Buffer ID, must be **`MFX_EXTBUFF_FEI_H265_PARAM`**.
`MaxCUSize`	Maximum size of coding units (CU) used by the encoder. It should be set to 16 or 32. If this value is 32, the FEI plugin will provide MV and distortion estimates for blocks up to size 32x32. Otherwise estimates are only provided for blocks up to size 16x16.
`MPMode`	Motion partition mode. Must be one of the following values: 1 – square partitions only 2 – square and symmetric rectangular partitions 3 – all modes supported, including asymmetric partitions If mode 2 or 3 is selected, MV and distortion estimates are provided for block sizes 8x16 and 16x8. If **MaxCUSize** is also >= 32, estimates are provided for sizes 32x16 and 16x32 as well. Otherwise these sizes are not evaluated.
`NumIntraModes`	Number of intra prediction modes to calculate. For each frame, the plugin returns a sorted list of the top **`NumIntraModes`** candidates for intra prediction. A unique set of modes is calculated for every block on a 4x4, 8x8,

	16x16, and 32x32 grid. (NOTE: currently this must be set to 1 – see "Known Limitations" above)
`reserved`	must be set to zero

Change History

This structure is available since SDK API 1.13.

mfxExtFEIH265Input

Definition

```
typedef struct
{
    mfxExtBuffer       Header;

    mfxU32             FEIOp;
    mfxU32             FrameType;
    mfxU32             RefIdx;

    mfxU16             reserved[22];
} mfxExtFEIH265Input;
```

Description

This structure should be passed with each call to **`MFXVideoEnc_ProcessFrameAsync`**. It specifies the type of processing to perform, frame type, and reference frame index (for correlating with the output.

Members

`Header.BufferId`	Buffer ID, must be **`MFX_EXTBUFF_FEI_H265_INPUT`**.
`FEIOp`	Operation to perform in the next call to **`MFXVideoEnc_ProcessFrameAsync`**, selected from the **`mfxFEIH265Operation`** enumerated list.
`FrameType`	Type of source frame (`MFX_FRAMETYPE_I`, `MFX_FRAMETYPE_P`, or `MFX_FRAMETYPE_B`).
`RefIdx`	Reference frame index to associate with the results of inter-frame prediction. This index will be used to access the corresponding output data when all processing is complete. This value must be less than the maximum number of reference frames specified during init (**`mfxVideoParam.mfx.FrameInfo.NumRefFrames`**).
`reserved`	must be set to zero

Change History

This structure is available since SDK API 1.13.

mfxExtFEIH265Output

Definition

```
typedef struct
{
    mfxExtBuffer       Header;

    mfxFEIH265Output   *feiOut;

    mfxU16             reserved[24];
} mfxExtFEIH265Output;
```

Description

This structure should be passed with each call to **MFXVideoEnc_ProcessFrameAsync**. It contains a pointer to the buffer which will receive output data from H265 FEI processing.

Members

`Header.BufferId`	Buffer ID, must be **MFX_EXTBUFF_FEI_H265_OUTPUT**.
`feiOut`	Pointer to user-allocated structure which will receive output data from H265 FEI processing.
`reserved`	must be set to zero

Change History

This structure is available since SDK API 1.13.

mfxFEIH265Output

Definition

```
typedef struct
{
    mfxU32                PaddedWidth;
    mfxU32                PaddedHeight;

    mfxU32                IntraMaxModes;
    mfxU32              * IntraModes4x4;
    mfxU32              * IntraModes8x8;
```

```
    mfxU32               * IntraModes16x16;
    mfxU32               * IntraModes32x32;
    mfxU32                 IntraPitch4x4;
    mfxU32                 IntraPitch8x8;
    mfxU32                 IntraPitch16x16;
    mfxU32                 IntraPitch32x32;

    mfxI32                 IntraPitch;
    mfxFEIH265IntraDist  * IntraDist;

    mfxI32                 PitchDist[64];
    mfxI32                 PitchMV[64];
    mfxU32               * Dist[16][64];
    mfxI16Pair           * MV[16][64];

    mfxI32                 InterpolateWidth;
    mfxI32                 InterpolateHeight;
    mfxI32                 InterpolatePitch;
    mfxU8                * Interp[16][3];

} mfxFEIH265Output;
```

Description

This structure contains the results of all H265 FEI processing for a source frame. It should be attached to the structure **mfxExtFEIH265Output**. A subset of the elements may be filled by a given call to **MFXVideoEnc_ProcessFrameAsync**, as determined by the value of **FEIOp**. The results may only be accessed after the corresponding call to **MFXVideoCORE_SyncOperation** has returned. For example, if MFX_FEI_H265_OP_INTRA_DIST is specified, then **IntraPitch** and **IntraDist[]** may be read once the corresponding sync operation has completed.

Members

PaddedWidth	Frame width padded up to a multiple of 16 if necessary. Used to access the output data in **IntraModes** and **IntraDist**.
PaddedHeight	Frame height padded up to a multiple of 16 if necessary.
IntraMaxModes	Maximum number of intra prediction modes returned (specified at initialization).
IntraModes4x4	Array containing **IntraMaxModes** candidate modes for each block of the input frame, on a 4x4 grid. The list is sorted with the best candidate appearing first. Modes are in the range [2,34] corresponding to the 33 directional modes in the HEVC standard. Modes 0 and 1 (planar and DC) are not considered. In memory, each block contains **IntraMaxModes** modes, so the modes for a 4x4 block with its upper-left pixel at location (x,y) would be accessed at:

	`(mfxU32)mode[i] =` **`IntraModes4x4`**`[`**`IntraMaxModes`**`*(y/4*`**`PaddedWidth`**`/4+x/4) + i];` for i = [0, **`IntraMaxModes`**)
`IntraModes8x8`	Array containing **`IntraMaxModes`** candidate modes for each block of the input frame, on an 8x8 grid. Addressing is similar to the 4x4 case.
`IntraModes16x16`	Array containing **`IntraMaxModes`** candidate modes for each block of the input frame, on an 16x16 grid. Addressing is similar to the 4x4 case.
`IntraModes32x32`	Array containing **`IntraMaxModes`** candidate modes for each block of the input frame, on a 32x32 grid. Addressing is similar to the 4x4 case.
`IntraPitch`	Horizontal pitch of the distortion estimates for intra coding on a 16x16 grid. In memory, each block is represented by a single **`mfxFEIH265IntraDist`** element, so the estimate for 16x16 block with its upper-left pixel at location (x,y) would be accessed at: **`IntraDist`**`[y/16*`**`IntraPitch`**` + x/16]`
`IntraDist`	Array containing the distortion estimates for intra coding.
`PitchDist`	Horizontal pitch of the distortion estimates for inter coding, one for each supported block size (partition unit).
`PitchMV`	Horizontal pitch of the estimated motion vectors for inter coding, one for each supported block size (partition unit).
`Dist`	Array containing the distortion estimates (SAD's) for inter coding for each source/reference frame pair. For `j = refIdx` and `k = MFX_FEI_H265_BLK_WxH` (block size index), `mfxU32 *`**`Dist`**`[j][k]` is the base pointer for distorition estimates for reference frame `j` on a `W` x `H` grid. **`PitchDist`**`[k]` returns the pitch of this array. For block sizes 16x16 and smaller, a single distortion value is provided, corresponding to the matching (quarter-pel) motion vector in MV. For block sizes 32x32, 16x32, and 32x16, a total of 9 distortion values are provided, corresponding to a 3x3 matrix centered around the returned value in MV (which has half-pel resolution). This allows the encoder to consider eight additional MV's with offsets of +/1 quarter-pel in each direction. See example code in sample_h265_gaa for an illustration of this indexing. Note: 16x32 and 32x16 blocks are only processed if **`mfxExtFEIH265Param.MPMode`** `> 1`, i.e. non-square blocks enabled.

MV	Array containing the motion vector estimates for inter coding blocks for each source/reference frame pair. Addressing is similar to **Dist**. See example code in sample_ h265_gaa.
InterpolateWidth	Width (in pixels) of the half-pel interpolated output frames. This includes a border of MFX_FEI_H265_INTERP_BORDER padding pixels.
InterpolateHeight	Height (in pixels) of the half-pel interpolated output frames. This includes a border of MFX_FEI_H265_INTERP_BORDER padding pixels.
InterpolatePitch	Pitch (in pixels) of the half-pel interpolated output frames.
Interp	Pointers to 8-bit Y planes containing interpolated frames. mfxU8 ***Interp**[j][k] corresponds to j = refIdx, and k = [0,1,2] where: 0 = horizontal interpolation (dx = 1/2, dy = 0) 1 = vertical interpolation (dx = 0, dy = 1/2) 2 = diagonal interpolation (dx = 1/2, dy = 1/2)

Change History

This structure is available since SDK API 1.13.

Enumerator Reference

mfxFEIH265BlockSize

Description

The `mfxFEIH265BlockSize` enumerator indicates the block size (partition units) for inter motion estimation. The convention is `MFX_FEI_H265_BLK_WxH`

Name/Description

`MFX_FEI_H265_BLK_32x32`	32x32 blocks
`MFX_FEI_H265_BLK_32x16`	32x16 blocks
`MFX_FEI_H265_BLK_16x32`	16x32 blocks
`MFX_FEI_H265_BLK_16x16`	16x16 blocks
`MFX_FEI_H265_BLK_16x8`	16x8 blocks
`MFX_FEI_H265_BLK_8x16`	8x16 blocks
`MFX_FEI_H265_BLK_8x8`	8x8 blocks

Change History

This enumerator is available since SDK API 1.13.

mfxFEIH265Operation

Description

The `mfxFEIH265Operation` enumerator indicates the type of operation(s) to perform in the call to **`MFXVideoEnc_ProcessFrameAsync`**. Multiple operations can be specified by OR'ing together more than one value.

Name/Description

`MFX_FEI_H265_OP_NOP`	no operation
`MFX_FEI_H265_OP_INTRA_MODE`	calculate best intra prediction mode(s)
`MFX_FEI_H265_OP_INTRA_DIST`	calculate distortion estimates for intra coding
`MFX_FEI_H265_OP_INTER_ME`	calculate motion vectors and distortion estimates for inter coding
`MFX_FEI_H265_OP_INTERPOLATE`	create half-pel interpolated frames

Change History

This enumerator is available since SDK API 1.13.