Intel® Media SDK 2014 Developer's Guide

Hardware Accelerated Video on Intel Platforms

LEGAL DISCLAIMER

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, the Intel logo, Intel Core 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.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

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 SSE3 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

This page left intentionally blank.

Contents

Revision History

The following table lists the revision schedule based on revision number and development stage of the product

Revision Number	Description	Date
1.0	Initial release of the document.	January, 2011
2.0	Updates to accompany Intel® Media SDK 2012 release	September, 2011
2.1	Updates accompanying Intel® Media SDK 2012 R2 release	April, 2012
3.0	Updates to accompany the Intel® Media SDK 2013 Release	December, 2012
3.1	Updates to accompany the Intel® Media SDK 2013 R2 Release	June, 2013
4.0	Updates to accompany the Intel® Media SDK 2014 Release	December, 2013

.

This page left intentionally blank.

1 About this Document

1.1 Overview

This document provides development hints and tips to help developers create the next generation of applications, enabling their end-users to have a great experience creating, editing, and consuming media content on Intel® Processor Graphics. Software development best practices are described using the latest revision of the Intel® Media SDK. The Intel Media SDK is free to download and free to use.

Before continuing with this document, be sure to download the latest version of this document and the Intel Media SDK from the product page: http://intel.com/software/mediasdk

This guide starts with how to understand the sample applications bundled with installation and expands from there to cover working with the Intel Media SDK from a software developer's perspective. It is intended to accompany the reference manuals

- mediasdk-man.pdf
- mediasdkmvc-man.pdf
- mediasdkusr-man.pdf
- mediasdkjpeg-man.pdf

to support rapid development of high performance media applications.

Chapter 1 covers documentation preliminaries.
Chapter 2 provides a high level overview of Intel Media SDK.
Chapter 3 covers installing Intel Media SDK and running the samples.
Chapter 4 provides an introduction to containers to simplify working with Intel Media SDK.
Chapter 5 provides a deep dive into Intel Media SDK application development.
Chapter 6 shows how to use Intel tools to improve performance.
The appendix contains additional information for specific usages.

1.2 Intended Audience

This document is intended for beginning and experienced media developers who are familiar with developing media applications on Microsoft* Windows*-based platforms.

1.3 Document Conventions, Symbols, and Terms

Table 1 *Coding Style and Symbols Used in this Document*

Source code:

```
// Initialize DECODE
decode->DecodeHeader(bitstream, InitParam_d);
decode->Init(InitParam_d);
```

Table 2 *Terms Used in this Document*

AAC	Advanced Audio Coding
API	Application Programming Interface
AVC1	Advanced Video Codec 1: H.264 Stream formatted without start codes
AVCHD*	AVCHD* Format, Version 1.00, Book 1: Playback System Basic Specifications
Blu-ray*	System Description Blu-ray* Disc Read-Only Format, Part 3, Audio Visual Basic Specifications, Version 2.2, December 2007
DirectShow*	Microsoft* DirectShow* Multimedia Framework
FFMPEG	Open Source video playback and conversion player
GPU	Graphics Processing Unit
H.264	ISO*/IEC* 14496-10 and ITU-T* H.264, MPEG-4 Part 10, Advanced Video Coding (AVC), May 2005
H.265/HEVC	High Efficiency Video Coding. ISO/IEC 23008-2 MPEG-H Part 2 and ITU-T H.265
HRD	Hypothetical Reference Decoder
IDR	Instantaneous decoding fresh picture, a term used in the H.264 specification
Intel® Processor Graphics	Intel® HD Graphics includes the latest generation of integrated graphics included in the same processor package of the Intel® microarchitecture codename Sandy Bridge, Ivy Bridge, and Haswell family of processors.
MFT	Microsoft Media Foundation* Transform
MPEG-2	ISO/IEC 13818-2 and ITU-T H.262, MPEG-2 Part 2, Information Technology- Generic Coding of Moving Pictures and Associate Audio Information: Video, 2000
SPS	Sequence Parameter Set
PPS NAL	Networked abstraction layer containing program

	parameter set
VC-1	SMPTE* 421M, SMPTE Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process, August 2005
VPP	Video Processing (implies both pre- and post-processing)

1.4 Related Information

Intel Graphics Developer's Guide: http://software.intel.com/en-us/articles/intel-graphics-developers-guides/

Intel Media SDK Support Forum: http://software.intel.com/en-us/forums/intel-media-sdk/

Intel Media SDK product website: http://www.intel.com/software/mediasdk

Intel Media SDK development tutorials: http://software.intel.com/en-us/articles/intel-media-sdk-tutorial

1.5 Specifications

The following table lists the specifications for the Intel® Media SDK 2014.

Table 3 *Intel® Media SDK 2014 Specifications*

Video Encoders	H.264 (AVC and MVC), MPEG-2, JPEG*/Motion JPEG, HEVC(SW)
Video Decoders	H.264 (AVC and MVC), MPEG-2, VC-1, JPEG*/Motion JPEG, HEVC(SW)
Video Processing Filters	Deinterlacing/Inverse Telecine, Resizing, Color Conversion, Denoising, Frame Rate Conversion, Brightness, Contrast, Hue, Saturation Control, Sharpening, Image Stabilization
Video Conferencing	Dynamic bitrate control, Low Latency, Error Detection/Resilience, temporal scalability, dynamic resolution change, long term reference frames, Rolling I-Frame
Extensions	User-defined filters (HEVC encoder/decoder delivered as plug-ins) Example plug-ins for OpenCL and VP8 decode

Supported OS	Microsoft Windows* 7 (32 and 64-bit) Microsoft Windows 8/8.1 (32 and 64-bit)
Supported Processors	Intel HD Graphics, 2nd, 3rd, and 4th generation Intel Core processor-based platforms, including Ultrabook™, a limited set of Intel® Xeon E3 processors, and Intel Atom™ processor-based tablets.

1.6 What's New?

1.6.1 New in Intel® Media SDK 2014

Highly Optimized for 4th Generation Intel® Core™ Processors

Support includes (not an exhaustive list):

- **HEVC SW Decode and Encode in standalone Intel Media SDK HEVC Software Pack**
- **Encode Enhancements**
  - Improved bit rate control for LookAhead bit rate control algorithm
  - Additional bit rate control algorithms such as Intelligent Constant Quality (ICQ)
  - Region of Interest encoding (ROI)
- **Video Processing Enhancement**
  - Frame Composition API
- **Audio Decode and Encode in standalone Intel Media SDK 2014 Audio Library:** AAC Decode & Encode, MP3 Decode
- **Container Splitter and Muxer API (sample):** MPEG-2 TS and MPEG-4
- **3rd party plug-in marketplace**

1.7 Accelerating video operations with the Intel® Media Software Development Kit

The Intel® Media Software Development Kit (Intel® Media SDK) abstracts the complex task of interacting with Intel® HD Graphics media acceleration interfaces to provide a high level view of video elementary stream processing operations. Since this enables highly optimized access to specialized hardware, it can be very fast and efficient. The high-level design also means that very good performance can be maintained as new generations of Intel® architectures emerge with few code changes. A software-only implementation is available with the same interface to provide compatibility with the widest range of processors. In this way the Intel Media SDK provides a portable and high-performance solution for the most demanding video processing tasks.

The Intel Media SDK optimized media libraries are built on top of Microsoft* DirectX*, DirectX Video Acceleration (DVXA) APIs, and platform graphics drivers. Intel Media SDK exposes the hardware acceleration features of Intel® Quick Sync Video built into 2nd, 3rd, and 4th generation Intel® Core™ processors. Read more at

http://www.intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html.

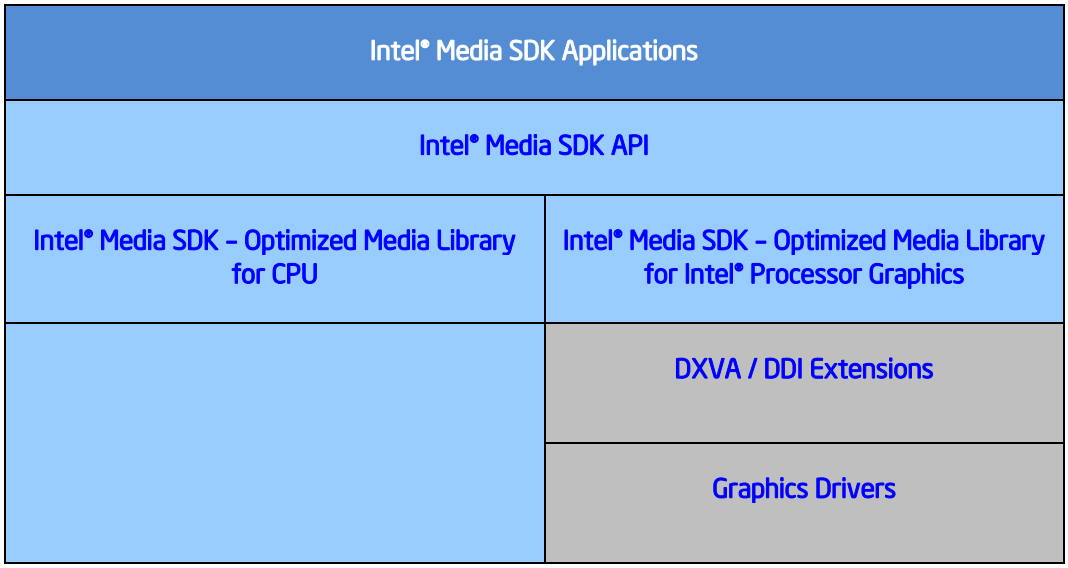


Figure 1 *Intel® Media SDK application software stack.*

While Intel Media SDK is designed to be a flexible solution for many media workloads, it focuses only on the media pipeline components which are commonly used and usually the most in need of acceleration. These are:

- Decoding from video elementary stream formats (H.264, MPEG-2, VC-1, and JPEG*/Motion JPEG, **new: HEVC**) to uncompressed frames
- Selected video frame processing operations
- Encoding uncompressed frames to elementary stream formats (H.264, MPEG-2, **new: HEVC**)
- **New**: Audio encode/decode and container split/muxing

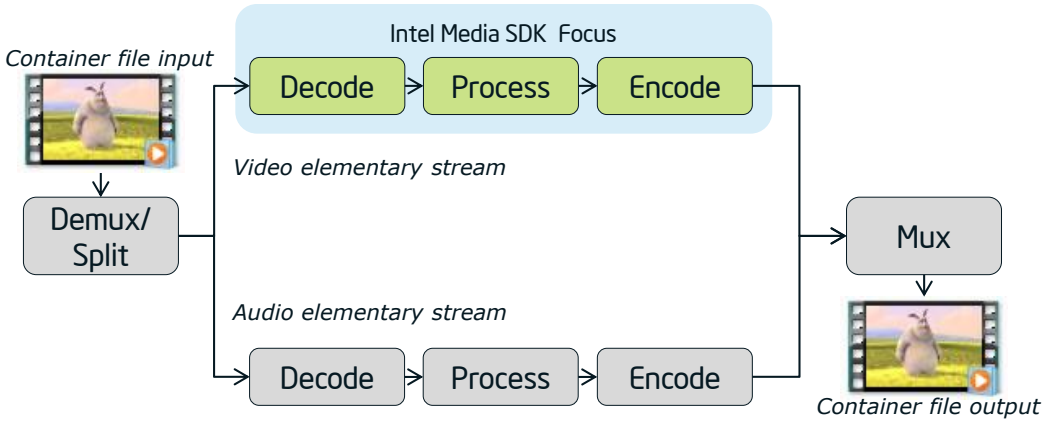


Figure 2 *A generic transcode pipeline. Intel® Media SDK accelerates a subset of the most computationally demanding video elementary stream tasks.*

Writing fully functional media applications capable of working with most media players requires that the application handle these operations. Media SDK 2014 now provides rudimentary audio codec features and container split/muxing functionality. An alternative is to leverage any of the open frameworks such as FFMPEG. A whitepaper is available showing this type of integration here: http://software.intel.com/en-us/articles/integrating-intel-media-sdk-with-ffmpeg-for-muxdemuxing-and-audio-encodedecode-usages. FFmpeg integration is also showcased as part of the Media SDK Tutorial: http://software.intel.com/en-us/articles/intel-media-sdk-tutorial

Intel Media SDK is designed to provide the fastest possible performance on Intel® Quick Sync Video hardware. This can be realized in traditional applications as well as Microsoft DirectShow* and Microsoft Media Foundation* Transform (MFT) plugins. While the complexities of the hardware accelerated implementation are hidden as much as possible, the API retains some characteristics of the architecture underneath which need to be accounted for in the design of any program using Intel Media SDK. These include asynchronous operation, NV12 color format, and Intel Media SDK's memory management infrastructure.

The Intel Media SDK API is high-level to provide portability. This future-proofs development efforts. Applications need not be redesigned to take advantage of new processor features as they emerge. The software implementation is designed to fill in where hardware acceleration is not available.

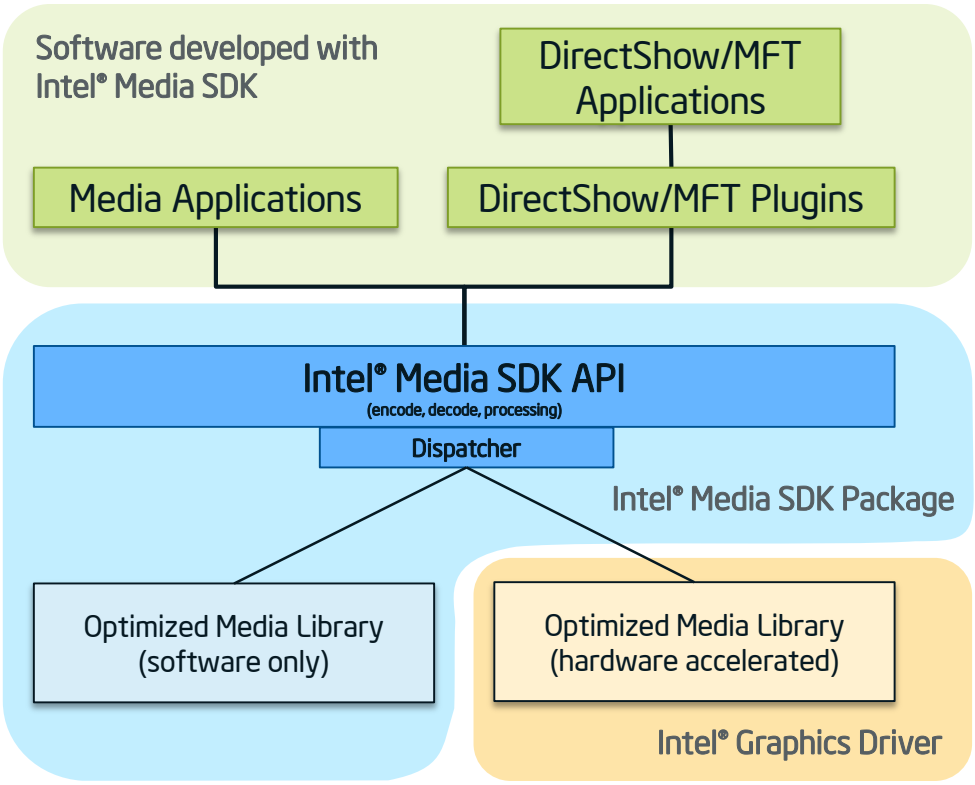


Figure 3 *Intel® Media SDK architecture showing software and hardware pathways*

Many platforms from Intel and other vendors do not have hardware acceleration capabilities. Intel Media SDK provides a software implementation using the same interface so its wide range of encoding, decoding, and pixel processing capabilities can be available for most platforms capable of running the supported operating systems. Thus applications developed on systems without Intel® Quick Sync Video can run with hardware acceleration when moved to a system with Intel® HD Graphics, or Intel® Iris™ —without changing a line of code. Conversely, applications developed for 2nd, 3rd, or 4th generation Intel® Core™ processor-based machines will still work on other systems.

The optimized software version of the Intel Media SDK can be used as a standalone package. However, the speed advantages of Intel Media SDK come from the graphics hardware, and a fully functional Intel® Processor Graphics driver must be installed to utilize the underlying hardware acceleration capabilities. (The default driver shipped with the system may not contain all of the necessary pieces. Please start your installation with a graphics driver upgrade.) The hardware and software libraries provide identical syntax for application development, but results may be different. The software library runs entirely on the CPU, while the platform-specific libraries execute using the CPU and GPU via the DXVA / DDI interfaces. Which one provides the best performance and quality will depend on several variables such as application implementation, content, hardware, mix of CPU vs. GPU operations, etc.

The dispatcher, which selects the version of the library used by your program, is an important part of the Intel Media SDK architecture. For more information on the dispatcher, please see section 4.3.

1.8 Tips For Distributing Intel® Media SDK applications

This is an attempt to summarize some fundamentals about the license agreement. Please refer to the Intel® Media SDK EULA and license.txt for more details, as they are the authoritative documents.

In general, the code in the samples directory can be compiled and distributed freely for Gold releases. Beta releases have not finished validation and are made available for learning purposes only.

The Microsoft* DirectShow* and Microsoft Media Foundation* Transform (MFT) sample codec filters with source code are in this category. However, several binary-only utilities such as splitters and audio codecs are included in the installation for development convenience only and cannot be distributed.

The header files, sample executables, binary splitters/muxers, and manuals are classified as "developer tools". Your application documentation can instruct users to install the Intel Media SDK, but cannot include these items.

You may repackage the software DLL to enable a self-contained install. However, since the hardware DLL is included with the graphics driver, this update process should be left alone to ensure system stability. A better solution may be to check the API version of the hardware DLL in your program and offer hints to install/upgrade as needed. In general, the most recent graphics driver/hardware DLL is the best one to use.

1.9 Developing Microsoft* Windows 8/8.1 Store Applications

For developers looking to get the most out of their Windows* Store applications with Intel fixed-function video acceleration, the Intel Media SDK provides samples and support enabling a subset of functionality by providing a certified Microsoft Media Foundation Transform (MFT) within the actual graphics driver provided with Intel client platforms (15.31 version or later). Developers are encouraged to utilize "sample_win8ui_transcode" as a foundation on how to correctly use the MFT.

2 Installing Intel® Media SDK and running the samples

2.1 Understanding Intel® Media SDK Distribution Components

Unlocking the full capabilities of the Intel® Media SDK requires:

- A machine with suitable video processing hardware and OS (see specifications table 3 in section 1.5).
- A graphics driver including the Intel Media SDK hardware acceleration DLLs.

2.1.1 Supported Processors and Operating Systems

Only 3rd and 4th generation Intel® Core™ processor family-based platforms fully support the latest API versions.

For more information on Intel's processor numbers, see

http://www.intel.com/products/processor_number/about/core.htm

Only Microsoft* Windows* 7 and Microsoft Windows 8/8.1 operating systems are currently supported.

2.1.2 Installing the graphics driver and Intel® Media SDK

The Intel® Media SDK is distributed in two parts. The Intel Media SDK itself is a standalone package, which can be used on systems which do not provide hardware support. It cannot access hardware acceleration without the hardware DLL which is distributed with the graphics driver.

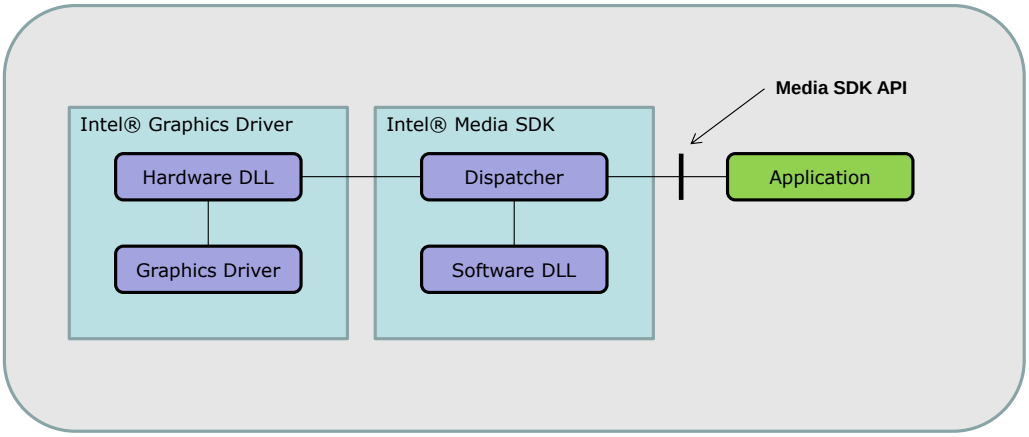


Figure 4 *Collaboration of graphics driver and Intel® Media SDK components*

Each component is separate. The Intel Media SDK can be installed before or after the driver, and the driver can be updated without re-installing Intel Media SDK. However, hardware acceleration is only available if both components are installed successfully.

Updating the graphics driver along with each update to the Intel Media SDK, if not more frequently, is highly recommended.

The media acceleration DLLs are distributed with the graphics driver, not the Intel Media SDK. In many cases the default graphics driver may not provide all of the files and registry entries needed. For best results, please update to the latest driver available from http://downloadcenter.intel.com/ before attempting a new install. While some systems may require a vendor-approved driver, the Intel drivers are usually appropriate. These drivers can be found by selecting "Graphics" as the product family at the Download Center site.

The graphics driver installer populates the Intel Media SDK directories under "c:\program files\Intel\Media SDK", or "c:\program files (x86)\Intel\Media SDK" on 64 bit OS installs. The Intel Media SDK hardware acceleration DLLs and certified HW Media Foundation Transforms can be found here.

2.1.3 Verifying successful installation

Successful installation of the graphics driver can be verified by checking for the \Program Files directories listed above and by checking the Intel Graphics and Media Control Panel. More information on accessing this panel can be found at

http://www.intel.com/support/graphics/sb/CS-022130.htm?wapkw=(graphics+and+media+control+panel)

The "Hello World" program in section 4.3.5 can be used as a quick check to verify that the Intel Media SDK and driver components are both installed and functional.

2.2 Using the Intel® Media SDK samples

2.2.1 Sample Overview

The Intel® Media SDK contains a rich set of samples to get a developer up to speed using the API quickly. Simple console applications and more complex samples leveraging some of today's most common media frameworks are provided for educational purposes. Additional documentation on the samples can also be found in:

- The readme in each sample package (build info and details for each sample)
- Intel Media SDK Filters Specifications (DLL info, pin interfaces, etc.)
- Intel Media SDK Sample Guide

Please note:

THESE ARE SAMPLES. THEY ARE INTENDED TO SIMPLIFY LEARNING THE IMPLEMENTATION OF THE INTEL® MEDIA SDK, NOT AS PRODUCTION-READY COMPONENTS.

2.2.2 Building the sample applications

Sample projects are created for Microsoft* Visual Studio* 2005. You will see a warning as part of the conversion with more recent versions of Microsoft Visual Studio. This is not a problem.

Each Media SDK sample is distributed as separate downloadable packages on the Media SDK portal web page.

2.2.3 Console Applications

Intel® Media SDK console applications demonstrate how to utilize the library without the additional complexity of a GUI or media framework. The "basic" samples are intended as a starting point for installation validation and new development. Each major component of the

SDK is represented by a basic console application as described in Table 4 *Intel® Media SDK Console Applications*.

Table 4 *Intel® Media SDK Console Applications*

Basic	sample_decode	Decoding from an elementary stream to raw (uncompressed) frames. Includes decoding of an elementary MVC (Multi-View Video Coding) video stream and use of the Stereoscopic 3D (S3D) API.
	sample_encode	Encoding from raw frames to a compressed (elementary) stream.
	sample_multi_transcode	Transcoding to and from elementary stream(s). Illustrates multiple asynchronous sessions to perform batch processing.
	sample_vpp	How to use pixel preprocessing to manipulate raw (uncompressed) frames.
Advanced	sample_full_transcode	Showcases a complete transcoding pipeline, including audio decode/encode and container splitting and muxing
	sample_user_modules	OpenCL™, VP8 decode user plugins.
	sample_utilities	Pipeline construction combining VPP and user plugins.
	sample_videoconf	Low latency, packet loss, dynamic bitrate, key frame insertion, long-term reference frame generation.

Console application samples design patterns

All of the samples share some common design characteristics.

- Pipelines: the output from one stage provides the input to the next. There can be multiple pipelines (such as in the multi transcode sample), but, as with the underlying Intel Media SDK implementation, there is an assumption of a linear series of "media building block" operations.
- Utilities: the sample_common directory contains utility classes for file I/O, surface/buffer allocation, etc. These are used by all of the samples. Additional functionality added here will be available to all samples.
- Opaque memory: Opaque memory is enabled for transcode pipelines, simplifying surface memory management.

2.2.4 Microsoft* DirectShow* and Microsoft* Media Foundation* Transform Samples

Microsoft* DirectShow* and Microsoft Media Foundation* transforms (MFT's) (HW accelerated MFTs are part of Intel HD graphics driver) are also provided to demonstrate how to use the Intel® Media SDK within these Microsoft media frameworks. To demonstrate the DirectShow Filters and/or Microsoft Media Foundation transforms, the Intel Media SDK also provides Microsoft Windows*-based GUI applications.

Repeating the sample disclaimer is especially important here. **These are samples only, not production-ready components,** even though they are accessed through the same Microsoft DirectShow/MFT interface as other production-ready filters.

These additional filters are distributed as binary only, and are "as is" with limited support. Why? These filters are part of the developer package to ensure that the GUI applications can create a complete Microsoft DirectShow graph. They are limited in functionality and are typically the source of developer frustration when using the Intel Media SDK to construct custom graphs. The internet offers alternatives to these filters, and developers are encouraged to experiment with other third-party filters if problems occur.

Table 5 *Microsoft* DirectShow* Filter Samples*

h264_dec_filter	Sample H.264 Decode filter
h264_enc_filter	Sample H.264 Encode filter
mpeg2_dec_filter	Sample MPEG-2 Decode filter
mpeg2_enc_filter	Sample MPEG-2 Encode filter
vc1_dec_filter	Sample VC-1 Decode Filter
mvc_dec_filter	Sample MVC Decode Filter
jpeg_dec_filter	Sample JPEG* Decode Filter

Table 6 *Windows*-Based GUI Applications*

DirectShow Player	Sample application that uses Microsoft DirectShow to decode and transcode media files
MediaFoundationTranscodeSample	Sample Media Foundation application showcasing how to utilize media transforms to achieve transcoding

2.2.5 Using the Framework Plugin Samples – Tips and Tricks

The Microsoft* DirectShow* and Microsoft Media Foundation* transforms (MFT) provide a new Intel Media SDK user with examples for using the API in real world scenarios. Even though the SDK provides GUI-based applications to leverage the sample filters, users can (and do) use the samples outside the context of the provided GUI applications. The GraphEdit* tool is quite often used to chain together DirectShow filters to create decode, encode, and/or transcode pipelines. This usage is encouraged. However, there are some tips that Intel Media SDK developers need to be aware of when working with these filters in standalone mode.

In addition to the filters and transforms listed in Table 5 *Microsoft* DirectShow* Filter Samples* the DirectShow sample package also contains the Microsoft DirectShow filters shown in Table 8. These are utility filters to help you get started. They are not redistributable.

Table 7 *Additional binary Microsoft* DirectShow* Filters*

imc_aac_dec_ds.dll	AAC Decoder
imc_aac_enc_ds.dll	AAC Encoder
imc_mp2_mux_ds.dll	MPEG-2 Multiplexer
imc_mp2_spl_ds.dll	MPEG-2 De-multiplexer (splitter)
imc_mp4_mux_ds.dll	MPEG-4 Multiplexer
imc_mp4_spl_ds.dll	MPEG-4 De-multiplexer (splitter)
imc_mpa_dec_ds.dll	MPEG-1 (MP3) Audio decoder
imc_mpa_enc_ds.dll	MPEG-1 (MP3) Audio encoder

Developers may find that the Intel Media SDK sample filters do not connect to some third-party filters. The developer package contains the supported interfaces of the Microsoft DirectShow filters listed in the "Intel Media SDK Filter Specifications" document. The most prevalent reason for the sample filters to refuse connection is that the color space is not supported. The Intel Media SDK sample filters require the input data to be in NV12 format. NV12 is the native format of the GPU, and thus the filters need this format to pass the data to the hardware for acceleration. Developers may find the need to insert a color conversion filter upstream of the sample decoder filter in order to connect the graph successfully. The Intel Media SDK does not provide a sample color conversion filter at this time. This is left to the developer to implement.

2.2.6 Intel Media SDK Tutorials

In addition to the included samples, developers new to the Intel Media SDK are encouraged to utilize the Intel Media SDK tutorials. This comprehensive curriculum is divided into seven distinct sections with increasing levels of complexity:

- Section 1: Introduces the Intel Media SDK session concept via a very simple sample.

- Section 2-4: Illustrates how to utilize the three core SDK components: Decode, Encode, and VPP.
- Section 5: Showcases transcode workloads, utilizing the components described in earlier sections.
- Section 6: Describes more advanced and compound usages of the SDK.
- Section 7: Explains how to integrate OpenCL processing into the Intel Media SDK pipelines.

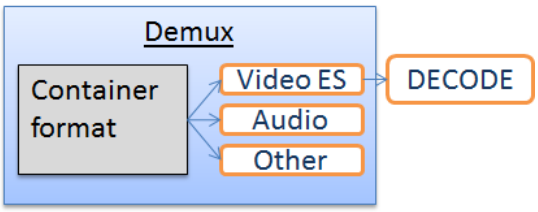
In addition to demonstrating how to implement the most common workloads, the tutorials also explain how to achieve optimal performance via a step-by-step approach. All tutorial samples are located in a self-contained Visual Studio* 2010 solution file. They can be downloaded here: http://software.intel.com/en-us/articles/intel-media-sdk-tutorial

3 Working with containers

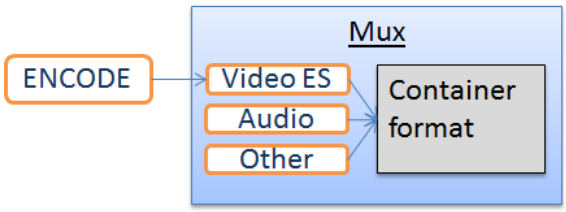
Disclaimer: Non-Intel tools are included for explanation only. Intel does not endorse or support them. It is up to you to determine if the license is appropriate for your situation.

The Intel® Media SDK is designed to reduce the complexities of decoding and encoding video elementary streams or containers (new for Media SDK 2014).

Typically, most source content is not in an elementary stream format. Rather, it is a combination of both video and audio "muxed" together into a single file via a container format. It must be demuxed to a video elementary stream before the Intel Media SDK decode can begin to work.



Many video players read only container formats and do not directly support the H.264 or MPEG-2 elementary streams output by Intel Media SDK encode. For these players, an additional muxing stage to a container format is required.



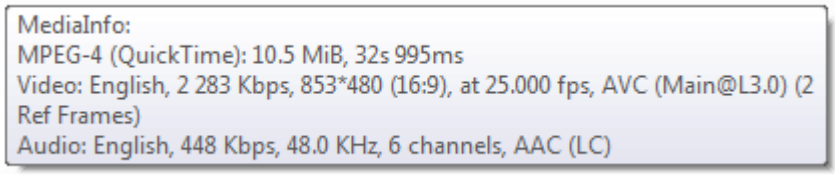
For players capable of working with elementary and/or raw formats, please see the Recommended Tools section.

3.1 Creating Elementary Streams from Existing Content

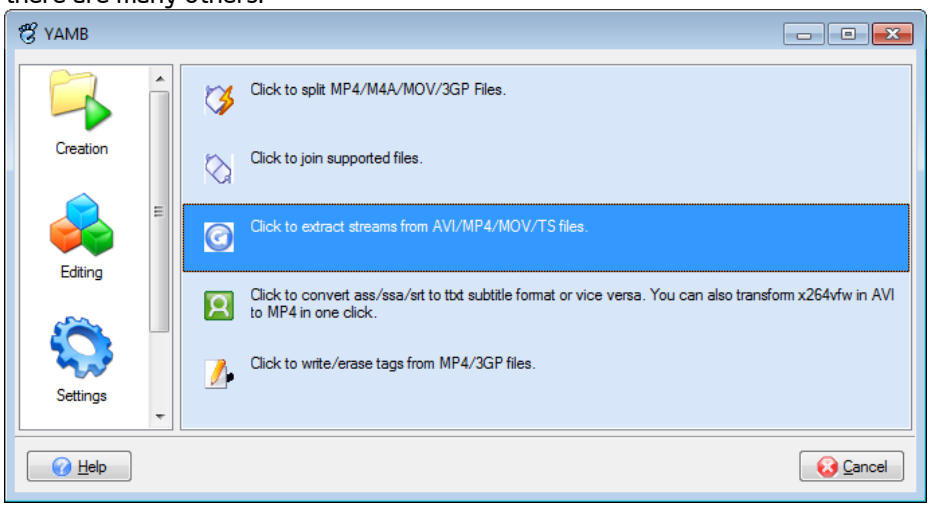
The Intel® Media SDK decoder and encoder components work with elementary streams. Most of the Media SDK console samples require an additional demuxing step as described above. Media SDK 2014 adds demuxing and muxing capability which can be utilized to extract/assemble media containers. There are also many free tools available to extract/assemble media containers.

The first step is obtaining some content to work with. The example in this section uses the free Big Buck Bunny trailer which can be obtained from www.bigbuckbunny.org.

There are many tools available to demux video streams from their containers. They are often specialized for a specific container type. It can be helpful to check what types of streams have been muxed together with a tool compatible with a wide variety of containers, such as MediaInfo. Here is MediaInfo's view of the streams in the Big Buck Bunny trailer (trailer_480p.mov):



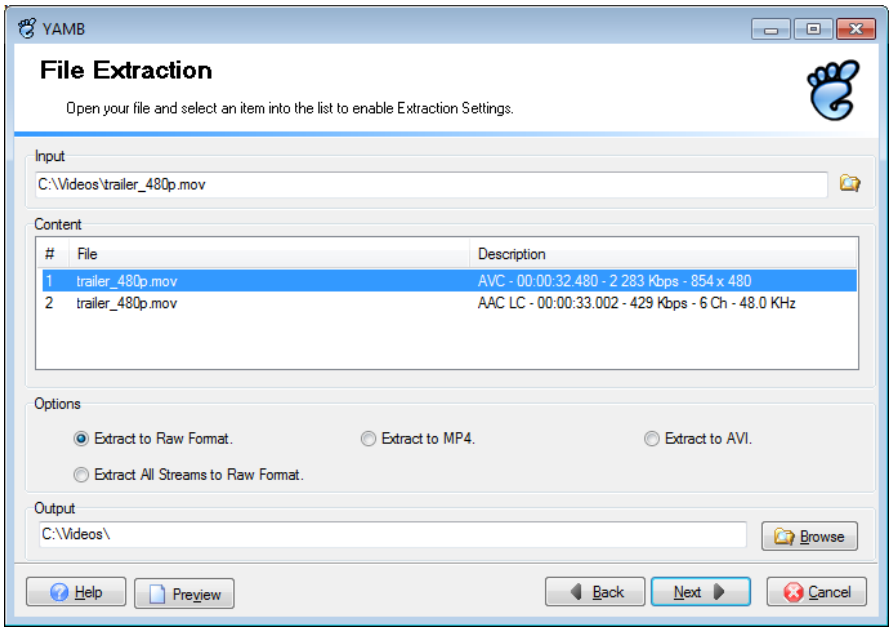
In this case the MPEG-4 container file holds 1 AVC/H.264 video stream and 1 audio stream. YAMB (Yet Another MP4Box Graphical Interface) is an easy to use tool for common tasks, but there are many others.



After selecting "Click to extract streams from AVI/MP4/MOV/TS files", enter the container file. The streams seen in MediaInfo should be available. Select the AVC stream and "Extract to Raw Format".

FFMpeg is also a good general purpose media tool:

C:\ ffmpeg -i <input.mp4 file> -vcodec copy -bsf h264_mp4toannexb -an -f h264 <output file for elementary stream>.h264



The output file (trailer_480p_track1.h264) is now in raw H.264 format, suitable for input to the Intel Media SDK decode and transcode console samples.

Other notable tools for this task are mp4creator and ffmpeg. Since they can be run from the command line they have the added advantage of easy automation in a script or build step. For example, this will demux the trailer as well:

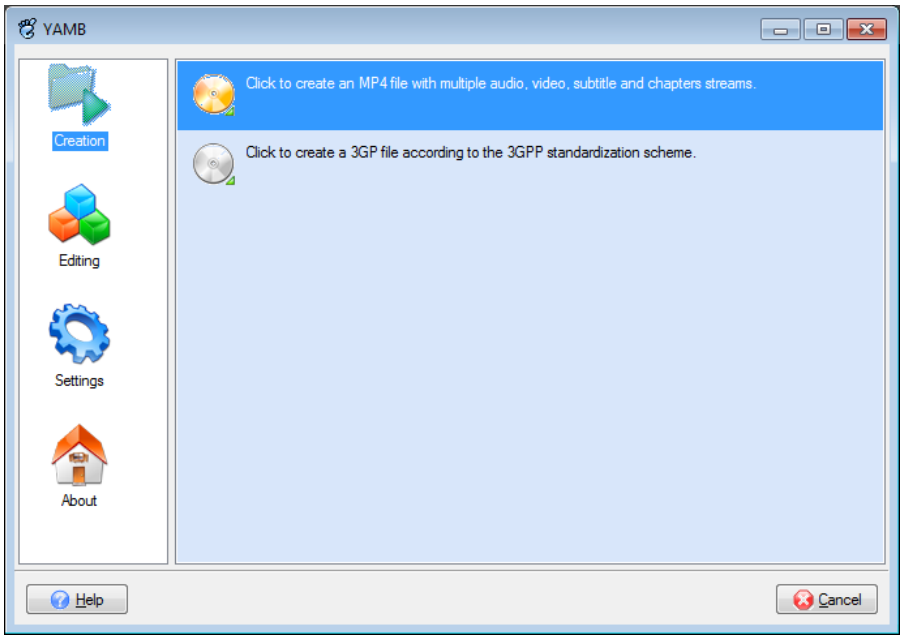
mp4creator –extract=1 trailer_480p.mov trailer_480p_t1.264

Several sites also host YUV sequences for encoder testing. These are packaged in many ways. The files at http://trace.eas.asu.edu/yuv/ are an example of the YUV format usable by the Intel Media SDK encoder console sample.

3.2 Creating Container Files from Elementary Streams

While it is possible to display output from MPEG-2 and H.264 files with elementary stream format compatible players such as mplayer, many video tools require a container format. The muxing process is similar to demuxing.

Here is another example with YAMB. First, select "Click to create an MP4 file..."



Just add the output file from the Intel® Media SDK encoder and the output file is ready to play.

The mp4creator and ffmpeg tools are also able to perform this muxing step. Since they are command line tools they are convenient to script or add to a Microsoft* Visual Studio* build step.

3.3 Muxing and Demuxing

Media SDK 2014 now supports the muxing and demuxing feature. We have also crafted a series of white papers on this topic (refer to links below) which details how to integrate mux/demux capabilities with Media SDK.

Using the Intel Media SDK with FFmpeg for mux/demuxing:

- http://software.intel.com/en-us/articles/integrating-intel-media-sdk-with-ffmpeg-for-muxdemuxing-and-audio-encodedecode-usages
- http://software.intel.com/en-us/articles/intel-media-sdk-tutorial-simple-6-transcode-opaque-async-ffmpeg

3.4 Decoder Bitstream Repositioning

Media SDK decoder bitstream repositioning is described in the Media SDK manual but the following information explains the concept in the context of container handling which is tightly connected to stream repositioning.

Please follow these steps to reposition a stream during a decoding session:

1. Invoke decoder MFXVideoDECODE_Reset() to reset the decoder
2. Clear current Media SDK bit stream buffer (mfxBitStream)
3. Reposition demuxer (splitter) to desired frame backward or forward in stream
   It is recommended that demuxer does the following before delivering frames to decoder:
   a. Reposition to closest I-frame to the desired position
   b. Insert sequence header (sequence parameter set for H.264, or sequence header for MPEG-2 and VC-1) into stream before the I-frame
      Note: Some streams may already contain sequence headers before each I-frame
   c. If sequence header is inserted before a frame that is not an I-frame, decoder may produce artifacts
4. Read data into Media SDK bit stream buffer from new demuxer position
5. Resume decoding by calling DecodeFrameAsync as usual.
   If the Media SDK decoder does not find any sequence headers while decoding from the new position DecodeFrameAsync will continuously return MFX_ERR_MORE_DATA (effectively asking for more bitstream data)

Media SDK encoder note: Encoder can control insertion of sequence headers via the "IdrInterval" parameter. Make sure that "GopPicSize" and "GopRefDist" values have been specified explicitly to ensure correct behavior. Also keep in mind that "IdrInterval" has slightly different behavior for H.264 vs. MPEG2 encoding. More details can be found in Media SDK manual.

4 Developing Intel® Media SDK Applications

4.1 Intel® Media SDK application design fundamentals

The Intel® Media SDK is designed to represent media operations as easy-to-use high-level building blocks. For example, decode takes a bit stream as input and produces surfaces as output. However, there are also some fundamental design characteristics to keep in mind to fully utilize the performance benefits of Intel Media SDK.

The list below outlines some basic architectural concepts for any Intel Media SDK project. The program's entire architecture does not need to fit these principles, but the section working with Intel Media SDK should have these characteristics:

- **Session/pipeline-based**: A session is created for a pipeline of steps. Surfaces may be shared internally between pipeline stages. Sessions use the dispatcher to map function calls to their DLL implementation.
- **Asynchronous**: each stage can have multiple frames "in flight", meaning that they are in some stage of processing on the CPU or GPU. In general, Intel Media SDK functions do not work like a traditional function call with a single input and output. Frame surface synchronization status needs to be checked. Frames may be locked while a session is working on them.
- **Based on the NV12 color format**: Decode/encode and VPP operations use NV12 because this provides better performance than other formats such as YUV. While it is possible to include color conversion filters it is best if pipelines can be arranged to minimize conversion steps by doing as much consecutive work in NV12 as possible.
- **Designed to minimize copies**: as with NV12 conversions, arrange pipeline steps to reuse surfaces in the same location instead of copying them between CPU and GPU.
- **Built with a Microsoft* Visual Studio* project/solution**.
- **Built around the Intel® Media SDK threading model**. The session's thread pool is responsible for Intel Media SDK's low level-tasks. Multiple sessions can be joined. In some cases pipelines can be parallelized with different models. This is particularly important if the software library is used.

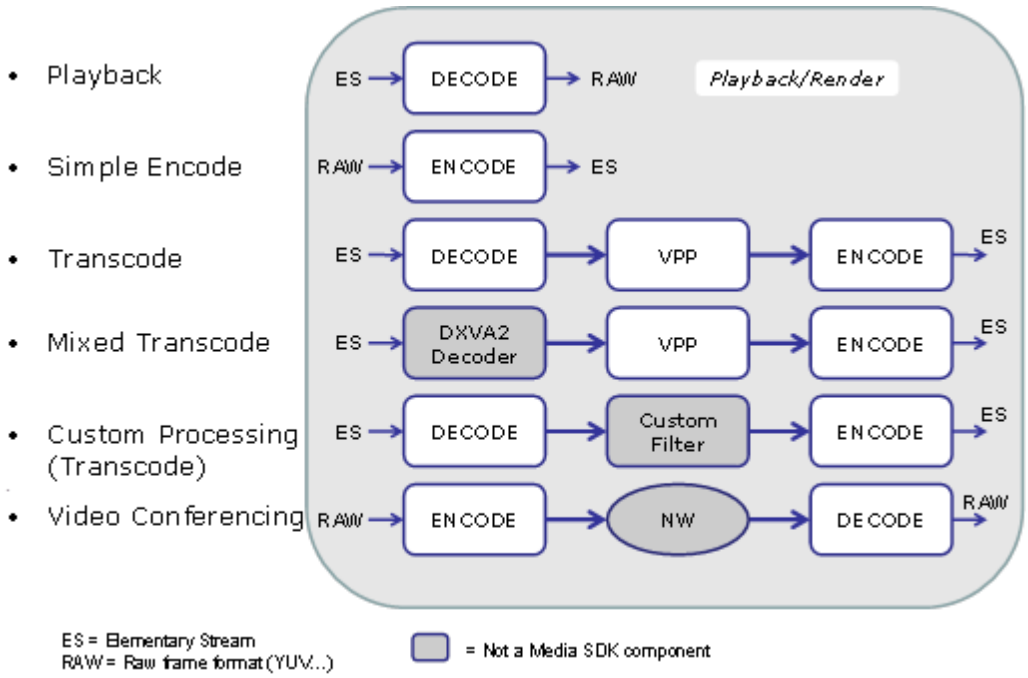
This is a general overview of what solutions will need to look like to get best performance. More information on performance is provided in Chapter 5.

4.2 Intel® Media SDK function groups

At its core, the Intel® Media SDK consists of five function groups accessed via the dispatcher. Syntax is identical in the software and hardware versions, though since they are different libraries, results can be different.

CORE	Auxiliary functions such as synchronization
DECODE	Decodes bitstreams into raw video frames
VPP	Processes raw video frames
ENCODE	Encodes raw video frames into bitstreams
USER	Performs user-defined algorithms as plugins

The Decode, VPP, Encode, and User function groups can be used as building blocks to provide the foundation for many different usage scenarios, such as:



While the manual is based on the C functions in mfxvideo.h, the samples use the C++ interface in mfxvideo++.h. In general the C++ functions are simply a wrapper around the corresponding C functions (see the appendix). The same function groups can be seen in both sets of functions.

4.3 Working with Intel® Media SDK Video Sessions

4.3.1 The dispatcher, software implementation, and software fallback

The media dispatching library, or dispatcher, is the gateway to Intel® Media SDK. This is statically linked to all Intel Media SDK-enabled programs. The dispatcher is responsible for exposing the entry points for the encoding, decoding, and video preprocessing routines. The

Dispatcher is also responsible for detecting and loading the appropriate implementation library for the client machine.

If the machine has compatible hardware and the dispatcher can find a way to access it (implying an appropriate graphics driver has been installed), the API calls will use the hardware implementation.

If software mode is requested, or Intel Media SDK is initialized in auto mode and no hardware implementation can be found, the software implementation will start.

Intel® Media SDK implementation	Description
MFX_IMPL_HARDWARE_ANY	Find the platform-specific implementation on any acceleration device
MFX_IMPL_HARDWARE(2-4)	Use the platform specific implementation of specific acceleration device
MFX_IMPL_SOFTWARE	Same interface as hardware implementation, but CPU only.
MFX_IMPL_AUTO	Dispatcher chooses hardware or software at runtime based on system capabilities.
MFX_IMPL_AUTO_ANY	Recommended to ensure processing on any hardware device with software fall back if needed.
Software Fallback	Same as above, but hardware unable to complete request. Uses same software implementation as MFX_IMPL_SOFTWARE but wrapped in the hardware DLL.

Use the following flags to specify the OS infrastructure that hardware acceleration should be based on:

MFX_IMPL_VIA_D3D9	Hardware acceleration via the Microsoft* DirectX 9 infrastructure
MFX_IMPL_VIA_D3D11	Hardware acceleration via the Microsoft* DirectX 11 infrastructure
MFX_IMPL_VIA_ANY	Hardware acceleration via any supported OS supported OS infrastructure

Please refer to the Media SDK reference manual and header files for details about additional implementation targets, primarily used to support systems with multiple GPUs.

The dispatcher determines which pathway to use at session startup. If the software implementation is chosen at startup, this will usually provide the greatest flexibility, though often not the best performance. If the hardware mode is chosen and the application cannot complete the operations requested by the parameters, Intel Media SDK goes into "software fallback" mode, which can work with a wider variety of inputs. Operations should be checked for the warning MFX_WRN_PARTIAL_ACCELERATION, which indicates that this has occurred.

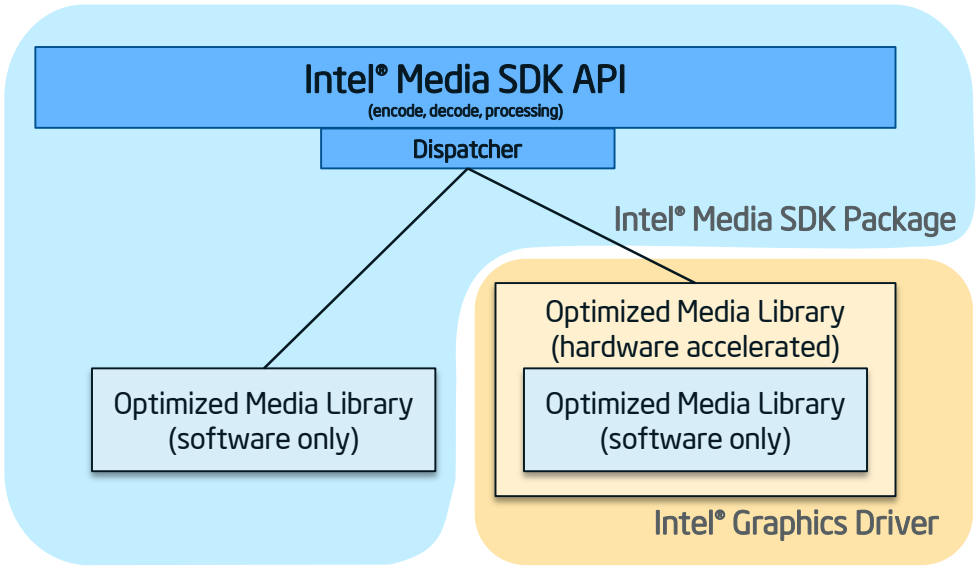


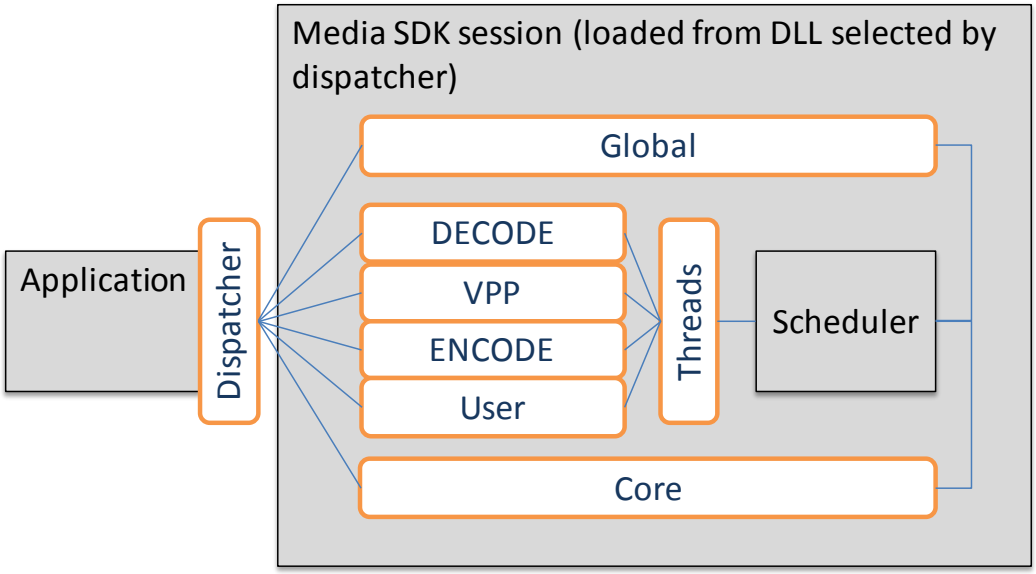
Figure 5 *The software implementation is also embedded in the hardware library to enable software fallback.*

Results from the software and hardware implementations can be different, in more ways than just speed. While the software implementation is a reference for the hardware accelerated version, in some cases different approaches are used if speed or quality can be improved.

The Media SDK dispatcher is also available as source code in the "opensource/mfx_dispatch" which is part of the Media SDK installer package.

4.3.2 Under the hood of the dispatcher

An application interfaces with the Intel® Media SDK through the Dispatcher library. The Dispatcher loads the appropriate library at runtime and sets up an SDK context called a "session" to the application. The session delivers the API's functionality and infrastructure (threads, schedule/sync capability, etc.); the dispatcher provides the interface.



The Dispatcher (libmfx.lib) must be statically linked to your application. It is responsible for locating and loading the correct library. In addition, it sets up the DirectX* context necessary for communicating with the GPU. This occurs even for software sessions using only system memory.

When an application initializes an Intel Media SDK session, the dispatcher will load either the software library (MFX_IMPL_SOFTWARE) or the hardware library appropriate for the platform (MFX_IMPL_HARDWARE).

To enable a solution capable of running on systems with or without hardware acceleration, the application can instruct the SDK to automatically choose the library with MFX_IMPL_AUTO. This is the recommended usage.

DLL Version	Search strategy
Hardware	"Common Files" locations for 32- and 64-bit DLLs specified in the registry by the graphics driver installer. DLL name includes codes for target platform. If a driver with the appropriate name is not found in the registry location used by the dispatcher, hardware session initialization will fail.
Software	Located by standard DLL search rules (i.e. system path). Intel Media SDK installer updates path with install target bin directory. If libmfxsw64.dll (libmfxsw32.dll for MFXInit run in 32 bit mode) cannot be found, the software session initialization will fail.

Figure 6 *Search strategy for hardware and software DLLs.*

Problems loading the hardware library could be related to version (e.g., a higher API version is available for software than hardware, which can happen in beta releases or if the graphics driver is not updated with a new Intel Media SDK install). Hardware acceleration problems can also often be traced to the driver. With MFX_IMPL_AUTO the software fallback happens silently, with no error. In many cases this is the desired functionality, though logging the version loaded can be helpful. MFXQueryIMPL and MFXQueryVersion can be useful to provide diagnostic feedback.

4.3.3 Intel® Media SDK sessions under the hood

Here are some fundamental ideas behind Intel® Media SDK sessions. Implementation details are subject to change, but there are several essential architectural components to be aware of:

- **Scheduler:** Manages subtask dependencies and priorities as multiple requests are handled asynchronously. The scheduler is responsible for synchronization, and reports the status code back when the sync point is queried.
- **Thread pool:** Started with session initialization to avoid thread creation overhead. The scheduler manages thread assignment to tasks.
- **Memory management core:** Intended to enable the highest possible performance via CPU and GPU allocation management, copy minimization, fast copy operations, atomic lock/unlock operations, opaque memory, etc.

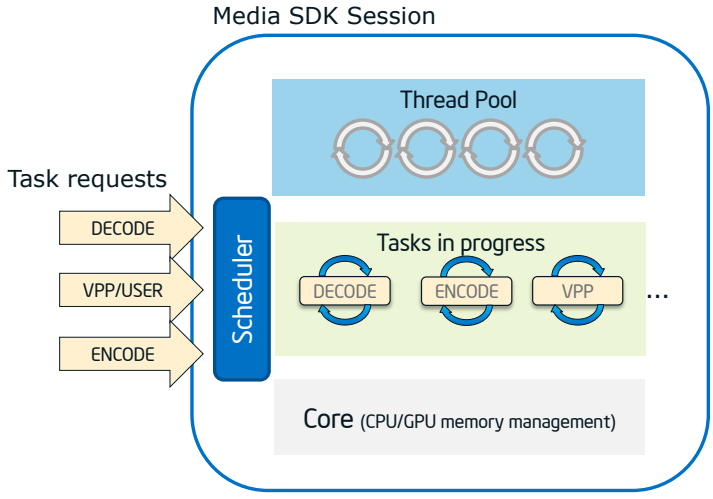


Figure 7 *Session internals*

All of these components work together to maximize asynchronous throughput of media tasks.

Since Intel Media SDK automatically creates several threads per session, this may interfere with other threading approaches, especially in software mode. Please see chapter 5 for more details on application performance optimization.

4.3.4 Creating Sessions

All applications must first create an Intel® Media SDK session to initialize the library and create a context for work to be done. Only one instance of the Decoder, Encoder, and VPP (preprocessing module) can run per session. Applications can create more than one session to accommodate multiple instances or pipelines. Software and hardware pipelines can exist simultaneously—this can be a way to balance GPU and CPU use for higher throughput.

To create a session, use the function:

mfxStatus sts = MFXInit(mfxIMPL impl, mfxVersion *ver, *session)

Parameters:

impl (in)	*Implementation requested (see below)*
ver (in)	API version requested (recommended: minimum API version for required functionality.)
session (out)	Session created.

The most commonly used implementations are:

MFX_IMPL_SOFTWARE	*Load the software library. No GPU acceleration.*
MFX_IMPL_HARDWARE	Load the platform-specific hardware library if available.
MFX_IMPL_AUTO	Attempt to load the hardware library, fall back to software if hardware is unavailable. More details in section 3.3.2.

Additional modes are provided for switchable graphics and multiple monitors. See Appendix D in the manual for more details. Note: there is an _ANY mode for the hardware and auto implementation types which extends support to scenarios with (or without) multiple graphics devices.

The available version numbers are:

SDK Version	Corresponding Intel® Media SDK release
1.0	Intel® Media SDK 1.5
1.1	Intel® Media SDK 2.0
1.3	Intel® Media SDK 2012

1.4	Intel® Media SDK 2012 R2
1.5	Intel® Media SDK for Clovertrail/Atom Platforms
1.6	Intel® Media SDK 2013
1.7	Intel® Media SDK 2013 R2
1.8	Intel® Media SDK 2014

SDK version is used for the version parameter of MFXInit, and is also used in the Intel Media SDK Reference Manual. Externally, Intel Media SDK is generally referenced by release (for example, Intel Media SDK 2012).

Intel Media SDK allows session initialization with an unspecified (null) version. This can be useful to determine the highest level of API supported by the loaded DLL (hardware or software). However, mismatches are possible when sessions are initialized with a null version:

- Beta releases: software implementation is released ahead of driver updates.
- Old graphics driver: The Intel Media SDK software implementation API version can be out of sync if an older graphics driver is on the system. Please update the graphics driver when updating Intel Media SDK—even more often is better.

Return status is either MFX_ERR_NONE (session is created and ready to use) or MFX_ERR_UNSUPPORTED (could not find the version/implementation requested). This status is very important to check, since most of the Intel Media SDK functions will not work without a valid session. Later return codes of MFX_ERR_INVALID_HANDLE may indicate an unusable session.

For most cases we recommend specifying a version instead of using the null version feature. The minimum version providing all functions used by the program is best, since this will allow the greatest portability. For example, even if API version 1.3 is available, it is probably not required if a session is only doing a simple encode—this can be accomplished by specifying version 1.0, which would enable portability even to machines with old drivers. However, null version can be an effective method of querying the age of the software and hardware DLLs for the purpose of informing users to update.

4.3.5 Query version and implementation

The following program is a minimal Intel® Media SDK "hello world". It attempts to start an Intel Media SDK software and hardware session. Once the session is active, its version and implementation can be queried.

```
#include "mfxvideo.h"
#include <stdio.h>

int main()
{
    mfxVersion SWversion = {0,1}, HWversion = {0,1}, version;
    mfxSession SWsession, HWsession;
    mfxStatus sts;

    sts = MFXInit(MFX_IMPL_SOFTWARE, &SWversion, &SWsession);
    if (MFX_ERR_NONE == sts)
    {
        MFXQueryVersion(SWsession,&version);
        printf("SW version:%d.%d  API level: %d.%d\n",
            SWversion.Major,SWversion.Minor,
            version.Major, version.Minor);
    }

    Sts = MFXInit(MFX_IMPL_HARDWARE, &HWversion, &HWsession);
    if (MFX_ERR_NONE == sts)
    {
        MFXQueryVersion(HWsession,&version);
        printf("HW version:%d.%d  API Level: %d.%d\n",
            HWversion.Major,HWversion.Minor,
            version.Major, version.Minor);
    }

    MFXClose(SWsession);
    MFXClose(HWSession);
}
```

Comparing the software and hardware API level can help determine if the graphics driver is out of sync and needs to be updated. Note: beta releases can offer a software "preview" of hardware capabilities, so a higher software API is expected.

Also note that there is a "mediasdk_sys_analyzer" tool distributed with the SDK package. The tool analyzes the developer platform and reports back Media SDK related capabilities.

4.3.6 Join/clone/disjoin session

Intel® Media SDK 2.0 (API 1.1) and later provides functionality to handle multiple simultaneous sessions/pipelines. This is designed to facilitate accurate system resource sharing to boost the performance of advanced workloads such as transcodes of multiple inputs at the same time.

An Intel Media SDK session holds the context of execution for a given task, and may contain only a single instance of **DECODE**, **VPP**, and **ENCODE**. Intel Media SDK's intrinsic parallelism, in terms of thread pool and scheduler, is well optimized for individual pipelines. For workloads with multiple simultaneous pipelines additional sessions are required. To avoid duplicating the resources needed for each session they can be "joined". This sharing also enables task coordination to get the job done in the most efficient way. Intel Media SDK 2.0 and later provide the following functions for multiple simultaneous sessions:

mfxCloneSession	After the initial session is created the application can create a child of the initial session. The application must complete all necessary pipeline initializations, including setting the allocator, initializing DECODE, VPP, and ENCODE using this new session. When created the cloned session has a separate task scheduler.
mfxJoinSession	The application then tells the SDK to share the task scheduler (thread pool) between parent and child sessions. Sharing the scheduler increases the performance by minimizing the number of threads, which in turn minimizes the number of context switches.
mfxDisjoinSession	After the batch conversion is complete, the application must disjoin all child sessions prior to closing the root session.
mfxClosesession	The application must also close the child session during de-initialization.

The performance impact with joined sessions is most noticeable when using the Intel Media SDK with software based sessions. This is because the CPU is prone to oversubscription. Joining sessions reduces the impact of this oversubscription.

With shared sessions, child session threads are placed under the control of the parent scheduler, and all child scheduler requests are forwarded. In addition to the performance benefits described above, this scheme enables coherent subtask dependency checking across the joined sessions.

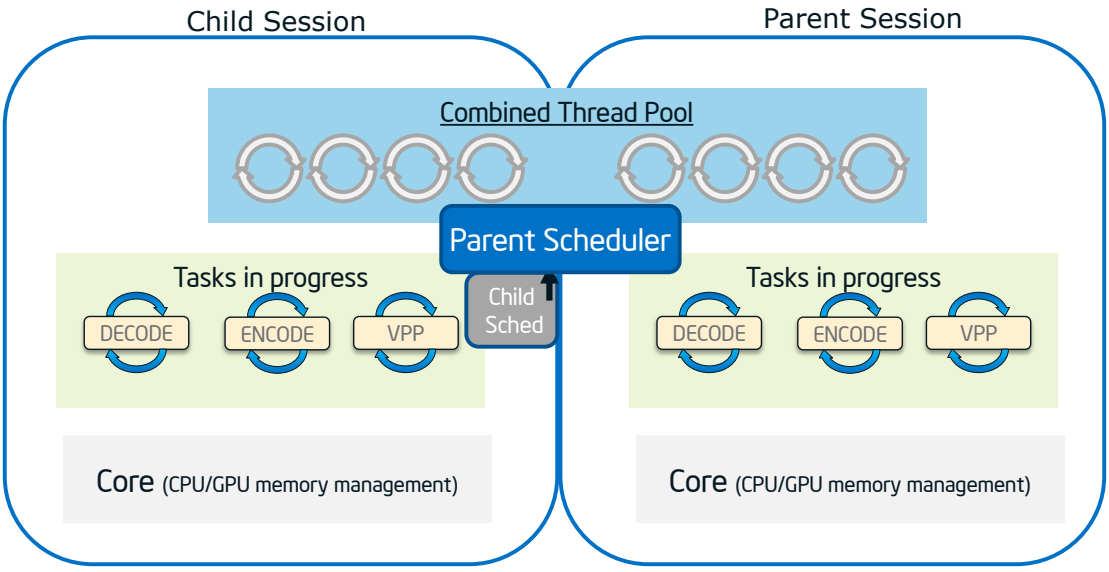


Figure 8 *A joined session, illustrating control forwarding from child scheduler.*

The relationship between parent and child sessions can be fine-tuned with priority settings.

4.4 Core Functions: Interacting with the asynchronous acceleration infrastructure

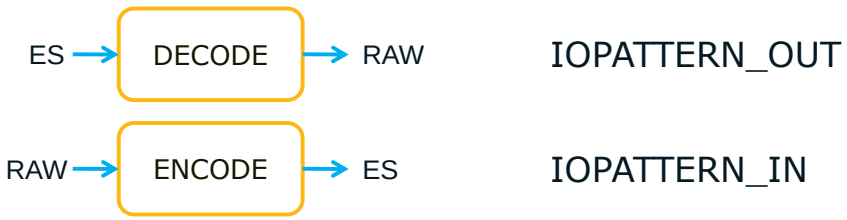
The Intel® Media SDK hides most of the complexity of efficiently pushing data through the session infrastructure, but awareness of the internal mechanisms can be helpful for developing applications.

- Surfaces need to be set up where the work will be done (CPU or GPU) to avoid extra copies.
- The session needs to buffer frames internally, as well as reorder them, to maximize performance.
- The session will estimate how many surfaces it needs. This needs to happen before work starts.
- As work is happening, some frame surfaces will be locked. The program controlling the session will frequently need to find an unlocked frame surface as it interacts with the session.

These steps are shared for all function groups.

4.4.1 IOPattern: System, Video, and Opaque memory

IOPattern specifies where frame data will be stored, and is a parameter for Intel® Media SDK stages.



Decode requires an IOPATTERN_OUT setting, encode requires IOPATTERN_IN.

SYSTEM_MEMORY specifies that CPU memory will be used. This is best for the software implementation.

VIDEO_MEMORY specifies that GPU memory will be used. This is best for the hardware implementation.

For transcode workloads, OPAQUE_MEMORY allows further simplification and optimization. Instead of requiring separate surface allocations to handle hardware and software implementation scenarios, the session manages the memory types and handoffs automatically.

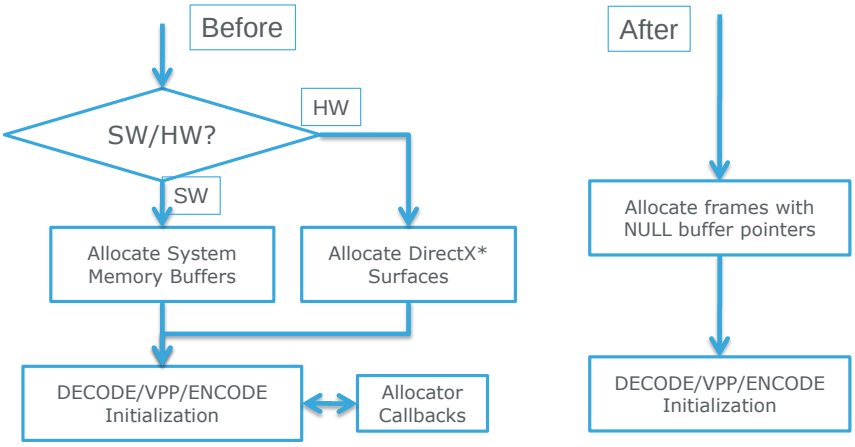


Figure 9: Code simplification with opaque surfaces

For an example of opaque surface use, please see sample_multi_transcode.

4.4.2 Surface reordering and locking

Intel® Media SDK reorders frames and builds reference lists to maximize performance. Extra surfaces are required to enable this mechanism. The goal is to minimize copies and use only "fast copies" where copies are required. Since this mechanism is operating concurrently with the main CPU program controlling the session, the session must make some surfaces unusable while it is working with them. This is to avoid race conditions, but also because many Intel optimizations are built on the assumption that surfaces will be locked while the GPU is writing. The controlling program must search for unlocked surfaces when interacting with a session.

The lock mechanism works much like other lock implementations. Internally, the Intel Media SDK session increases a lock count when a surface is used and decreases the count when done. Applications should consider the frame off limits when locked.

Pseudo code:

```
Allocate a pool of frame surfaces
decode->DecodeHeader()
decode->Init()
while (bitstream not finished)
{
        Locate an unlocked frame
        Fill the frame with encode input
        Send the frame to the SDK encoding function
        Sync
        Write any output bitstream
}
De-allocate frame surfaces
```

4.4.3 Surface Allocation with QueryIOSurf

For greatest efficiency all surfaces used should be allocated at session creation. The session needs enough surfaces to implement everything in its pipeline, as well as a few extra to accommodate asynchronous reordering and optimized data sharing between stages. To accomplish this, each pipeline stage (encode, decode, etc.) needs to call QueryIOSurf to determine how many surfaces will be needed for the specified configuration.

The application asks each stage how many surfaces are necessary. The total number of surfaces to allocate is the sum of all stages.

The Intel® Media SDK natively supports I/O from both system memory and Microsoft* Direct3D* surfaces. Generally speaking, the Intel Media SDK performs best when the memory type selected matches the way the SDK was initialized. For example, if an application initializes a session with MFX_IMPL_SOFTWARE, performance for that session is improved by using system memory buffers. SDK sessions initialized to use the hardware acceleration libraries (MFX_IMPL_HARDWARE) will benefit from the use of Microsoft Direct3D surfaces. Not using the correct memory type can cause significant performance penalties, because each frame of data must be copied to and from the specified surface type.

Applications can control the allocation of surfaces via the mfxBufferAllocator and mfxFrameAllocator interfaces. When these interfaces are not implemented, the Intel Media SDK allocates system memory by default. See the "Memory Allocation and External Allocators" section in the Intel Media SDK Reference Manual.

```
// calculate number of surfaces required for decoder
sts = m_pmfxDEC->QueryIOSurf(&m_mfxVideoParams, &Request);
MSDK_IGNORE_MFX_STS(sts, MFX_WRN_PARTIAL_ACCELERATION);
MSDK_CHECK_RESULT(sts, MFX_ERR_NONE, sts);

nSurfNum = MSDK_MAX(Request.NumFrameSuggested, 1);

// prepare allocation request
Request.NumFrameMin = nSurfNum;
Request.NumFrameSuggested = nSurfNum;
memcpy(&(Request.Info), &(m_mfxVideoParams.mfx.FrameInfo), sizeof(mfxFrameInfo));
Request.Type = MFX_MEMTYPE_EXTERNAL_FRAME | MFX_MEMTYPE_FROM_DECODE;

// add info about memory type to request
Request.Type |=
  m_bd3dAlloc ? MFX_MEMTYPE_VIDEO_MEMORY_DECODER_TARGET : MFX_MEMTYPE_SYSTEM_MEMORY;
// alloc frames for decoder
sts = m_pMFXAllocator->Alloc(m_pMFXAllocator->pthis, &Request, &m_mfxResponse);
MSDK_CHECK_RESULT(sts, MFX_ERR_NONE, sts);

// prepare mfxFrameSurface1 array for decoder
nSurfNum = m_mfxResponse.NumFrameActual;
m_pmfxSurfaces = new mfxFrameSurface1 [nSurfNum];
```

4.4.4 Finding unlocked buffers

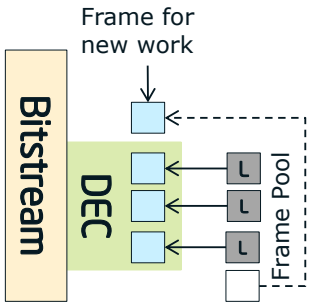
Since the application manages the surface pool, it must also be aware of which surfaces are currently in use by asynchronous operations. Locked surfaces cannot be used so a search must be done for an unused surface when additional work is added to an Intel® Media SDK stage.

```
while (MFX_ERR_NONE <= sts ||
       MFX_ERR_MORE_DATA == sts ||
       MFX_ERR_MORE_SURFACE == sts)
{
   // . . .
   else if (MFX_ERR_MORE_SURFACE == sts || MFX_ERR_NONE == sts)
   {
      // find new working surface
      nIndex = GetFreeSurfaceIndex(m_pmfxSurfaces, m_mfxResponse.NumFrameActual);
   }

   sts = m_pmfxDEC->DecodeFrameAsync(&m_mfxBS,
                                     &(m_pmfxSurfaces[nIndex]),
                                     &pmfxOutSurface,
                                     &syncp);
   // . . .
```

The surface search can be straightforward, as below.

```
mfxU16 GetFreeSurfaceIndex(mfxFrameSurface1* pSurfacesPool, mfxU16 nPoolSize)
{
    if (pSurfacesPool)
    {
        for (mfxU16 i = 0; i < nPoolSize; i++)
        {
            if (0 == pSurfacesPool[i].Data.Locked)
            {
                return i;
            }
        }
    }

    return MSDK_INVALID_SURF_IDX;
}
```

The following diagram illustrates the buffer search process. Individual members of the frame/surface pool are managed externally. Those currently in use are flagged as locked, so finding an unlocked surface requires the application to search for one that is available.



L=locked

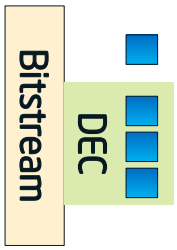
4.5 Decode overview

The following pseudo code illustrates the steps required for decode. Please note that this code is synchronous—better performance can be obtained from an asynchronous implementation.

```
set_up_params(&InitParam_d);
set_up_allocator(InitParam_d);

// Initialize DECODE
decode->DecodeHeader(bitstream, InitParam_d);
decode->Init(InitParam_d);

// Decode frames from stream
while (bitstream) {
              read_from_file(&bitstream);
              find_free_work_frame(&frame_w);
              decode->DecodeFrameAsync(bitstream, frame_w, frame_d, sync_d);
              SyncOperation(sync_d);
              write_frame(frame_d);
}

// Decode frames still waiting to drain
while (MFX_ERR_MORE_SURFACE == sts) {
              find_free_work_frame(&frame_w);
              decode->DecodeFrameAsync(NULL, frame_w, frame_d, sync_d);
              SyncOperation(sync_d);
              write_frame(frame_d);
}

// Close components
decode->Close();
delete_surfaces_and_allocator();
```

Here the decoder is initialized from the stream via DecodeHeader. Since operation is asynchronous, two loops are required.

The main asynchronous loop processes multiple frames at a time:



When the bitstream is done, frames may still be in the decoder, so an additional loop is required to drain them.

4.6 Encode overview

The following pseudo code illustrates the steps required to encode.

```
set_up_params(&InitParam_e);
set_up_allocator(InitParam_e);

// Initialize ENCODE
encode->Init(InitParam_e);

// Encode stream from frames
while (bitstream) {
       find_free_frame(&frame_e);
       load_frame(&frame_e);
       encode->EncodeFrameAsync(NULL, frame_e, bitstream, sync_e);
       SyncOperation(sync_e);
       write_bitstream(bitstream);
}

// Encoded bitstream still waiting to drain
while (MFX_ERR_MORE_SURFACE == sts) {
       encode->EncodeFrameAsync(NULL, NULL, bitstream, sync_e);
       SyncOperation(sync_e);
       write_bitstream(bitstream);
}

// Close components
encode->Close();
delete_surfaces_and_allocator();
```

4.7 Decode and Encode parameterer validation

The Encoder and Decoder components validate the set of parameters supplied but depending on the operation the scope of the validation differs. Please refer to the table below for further details on the behavior of each function.

Function	Behavior
DecodeHeader	Parses the input bit stream and populates mfxVideoParam structure. Parameters are NOT validated. Decoder may or may not be able to decode the stream
Query	Validates the input parameters in mfxVideoParam structure completely. Returns the corrected parameters (if any) or MFX_ERR_UNSUPPORTED if parameters cannot be corrected. Parameters set as zero are not validated.
QueryIOSurf	Does NOT validate the mfxVideoParam input parameters except those used in calculating the number of input surfaces
Init	Validates the input parameters in mfxVideoParam structure completely. Some parameters may be corrected if selected configuration is not compatible. If compatibility cannot be resolved MFX_ERR_INVALID_VIDEO_PARAM is returned. Note: GetVideoParam() may be used to retrieve the corrected parameter set

4.8 Encode Bitstream Buffer Allocation with GetVideoParam

The MFXEncode_GetVideoParam function provides a way to determine the Encoder's working parameter set. One of the values the function returns is the size of the bit stream buffer needed for the workload, BufferSizeInKB. Use this function prior to bitstream allocation; it returns the minimal buffer size required for most scenarios. It is possible to encounter an increase in the required minimum buffer allocation. When this occurs, the Encoder returns MFX_ERR_NOT_ENOUGH_BUFFER. If this situation arises, call MFXEncode_GetVideoParam again to retrieve the new allocation size, reallocate the correct amount of buffers, and then rerun **ENCODE** with the new sizes.

4.9 Encode Quality and Performance settings

The Intel Media SDK predefines various target usages (TU) based on a range of different performance and quality tradeoffs. The Media SDK API allows developers to configure speed vs. quality to refine the encoder to match their usage models. The default TU setting is "Balanced", and is set via the mfxInfoMFX.TargetUsage parameter.

Best Quality ←----------→ Best Performance

Target Usage	TU1	TU2	TU3	TU4	TU5	TU6	TU7

While the Media SDK API always supported seven distinct target usages, in general the API mapped the settings into three distinct enumerators: MFX_TARGETUSAGE_BEST_QUALITY, MFX_TARGETUSAGE_BALANCED, and MFX_TARGETUSAGE_BEST_SPEED.

The introduction of Intel® Iris™ Pro Graphics, Intel® Iris™ Graphics and Intel® HD Graphics (4200+ Series) has enabled the Intel Media SDK to evolve the use of this setting by expanding it to include seven distinct target usages, each with its own particular balance of quality and performance. The refined granularity of the TU settings enables the Media SDK library to introduce specific algorithms for a particular setting. The Intel Media SDK 2013 R2 includes the first of these TU specific optimizations: Look Ahead.

4.9.1 Rate Control Methods

There are several custom bit rate control modes in the Media SDK AVC encoder that has been designed to improve encoding quality and behavior for specific usage models. Some of these use a technique known as Look Ahead (LA). The technique produces especially good results on video sequences with static to fast motion scene changes. The new algorithm works by performing extensive analysis of the requested number of future frames before encoding. The number of frames examined is set by the `LookAheadDepth` member of the `mfxExtCodingOption2` structure. LA estimates the complexity of frames, relative motion, and inter frame dependencies to distribute the bits across frames to yield the best quality. The new algorithm works with any GOP pattern, and the presence of B frames yield the largest quality enhancement.

The Look Ahead enhancements do carry a cost. Besides increased latency, there is increased memory consumption because the frames being analyzed must be stored. The amount of memory used is directly related to the `LookAheadDepth`. LA may also increase the encode execution time. The analysis work is done asynchronously to the actual encoding and the encoding step is completed once the desired parameters are determined. The performance impact for is ~20% using TU7 and for TU1-4 the look ahead performance impact compared to VBR mode is negligible. Of course, developers are encouraged to experiment and make a determination on what is best for their application.

A publicly available whitepaper for the `MFX_RATECONTROL_LA` method is available that discuss the tradeoffs in deeper detail:
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CC4QFjAA&url=http%3A%2F%2Fnewsroom.intel.com%2Fservlet%2FJiveServlet%2FdownloadBody%2F3880-102-1-6896%2FWP-HD-Graphics-4200.pdf&ei=RnaWUqKWMsn6oASdo4CIDQ&usg=AFQjCNGdlhifenj9Bn522Rk519M6wmsZhw&sig2=qCLh4xazBNBr2g-cGUac8w&bvm=bv.57155469,d.cGU

4.9.1.1 Enabling Look Ahead

To enable Look Ahead, an application must set the mfxInfoMFX::Rate Control to `MFX_RATECONTROL_LA` or `MFX_RATECONTROL_LA_CRF`. For both algorithms the InitialDelayInKB and MaxKbps parameters are ignored. For `MFX_RATECONTROL_LA`, the only

available rate control parameter is mfxInfoMFX::TargetKbps, and for `MFX_RATECONTROL_LA_CRF` the parameter mfxInfoMFX::CRFQuality is used.

The amount of frames the encoder analyzes controlled by a developer. Set the look ahead depth via the mfxExtCodingOption2::LookAheadDepth parameter. Valid values are from 10 to 100 inclusive. The default (and recommended) value is 40, which can be set by setting this parameter to 0.

There are some limitations to LA that need to be noted for the Intel Media SDK 2014 release:

- Look Ahead is only available on Intel® Iris™ Pro Graphics, Intel® Iris™ Graphics and Intel® HD Graphics (4200+ Series)
- Intel Media SDK must be initialized to use API version 1.7 or newer for `MFX_RATECONTROL_LA`, and 1.8 or newer for `MFX_RATECONTROL_LA_CRF`.
- Only progressive content is supported at this time
- In certain circumstances HRD requirements may be violated

4.10 Mandatory VPP Filters

The Intel® Media SDK includes commonly used video processing operations: color space conversion, scaling, deinterlacing, and frame rate conversion. These are also called "mandatory" filters since they must be available for all Intel Media SDK implementations.

These core filters are triggered by any inconsistencies between the mfxVideoParam struct parameters (as listed in Figure 9) for input and output surfaces. The input and output parameters are compared and filters are switched on or off as required. If there is a difference requiring a filter to change the frame, the filter will run. If there are no differences in the affected parameters, the filter does not run. While Intel Media SDK can minimize the overhead of additional pipeline stages and much of the work can be done in parallel, there may be a significant performance cost associated with adding additional steps.

The following pseudo code illustrates the steps required for VPP processing. These are very similar to encode and decode.


```
// Initialize
vpp->Init(InitParam_v);
// run VPP for all frames in input
while (frame_in) {
       vpp->RunFrameVPPAsync(frame_in, frame_out, sync_v);
       SyncOperation(sync_v);
}

//drain loop
while (buffered data remains) {
       vpp->RunFrameVPPAsync(NULL, frame_out, sync_v);
       SyncOperation(sync_v);
}

// Close components
vpp->Close();
```

VPP Filter	Parameters checked	Description
Surface format conversion	ColorFourCC, ChromaFormat	Execute conversion filter to match output color format.
Deinterlace/Inverse Telecine	PicStruct	Convert interlaced input to progressive output.
Resize/crop	Width, Height, Crop{X,Y,W,H}	If there is a difference in frame size or crop settings between input and output frames, run the resize stage.
Frame rate conversion	FrameRateExtN, FrameRateExtD	Support arbitrary input/output frame rate conversion based on simple frame dropping / repetition.

Figure 11 *Mandated VPP Filters*

4.10.1 Surface format conversion

Intel® Media SDK uses NV12 as its native format. It is best to minimize color conversions by arranging pipelines for a single conversion at the beginning and end of Intel Media SDK processing.

This filter handles conversions between FourCC color formats (RGB32, YUV/YV12, YUY2 to NV12).

4.10.2 Tips for Deinterlacing Content

VPP implements deinterlacing by either specifying input and output formats for the entire incoming image sequence, or by explicitly setting the parameter for every input frame. To do this, use the PicStruct parameter from mfxFrameInfo.

When the input is interleaved and the output is progressive, the Intel® Media SDK enables deinterlacing. If the application cannot detect the input picture structure, set PicStruct to MFX_PICSTRUCT_UNKNOWN for the input format at VPP initialization, and provide the correct input frame picture structure type on every frame.

VPP is generally nested in the pipeline after **DECODE**. In this case, use the output parameters of the decoder to initialize and process VPP frames.

The following pseudo code example demonstrates how to configure the SDK VPP with parameters from **decode**:

```
sts = pmfxDEC->DecodeHeader(pmfxBS, &mfxDecParams);

if (MFX_ERR_NONE == sts) {
    memcpy(&mfxVppParams.vpp.In, &mfxDecParams.mfx, sizeof(mfxFrameInfo));
    memcpy(&mfxVppParams.vpp.Out, &mfxVppParams.vpp.In, sizeof(mfxFrameInfo));
    mfxVppParams.vpp.Out.PicStruct = MFX_PICSTRUCT_PROGRESSIVE;
}
...
sts = m_pmfxDEC->DecodeFrameAsync(pmfxBS, pmfxSurfaceDecIn,
                                  &pmfxSurfaceDecOut, &mfxSyncpDec);
if (MFX_ERR_NONE == sts) {
    m_pmfxVPP->RunFrameVPPAsync(pmfxSurfaceDecOut, pmfxSurfaceVPPOut,
                                NULL, &mfxSyncpVPP);
}
```

4.10.3 VPP Scaling

Scaling is one of the most frequently used VPP algorithms. It generally can contain several operations: cropping, resizing, and stretching together with keeping the aspect ratio intact.

To perform scaling, define the region of interest using the CropX, CropY, CropW and CropH parameters to specify the input and output VPP parameters in the mfxVideoParam structure. These parameters should be specified per frame.

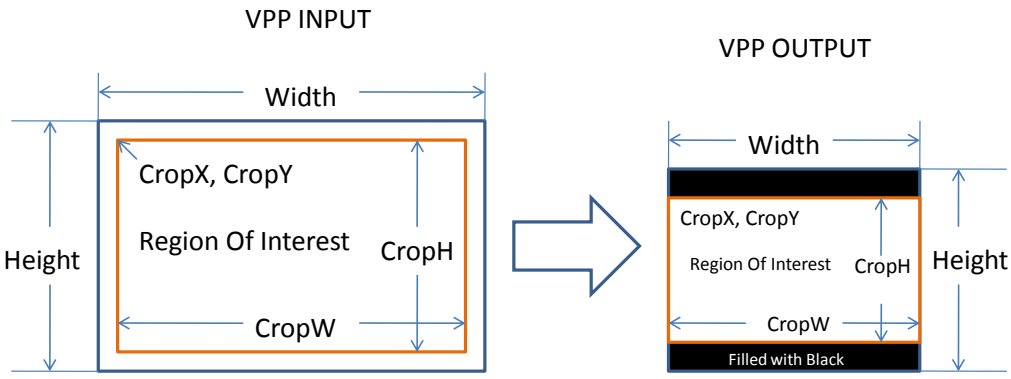


Figure 5 *VPP Region of Interest Operation*

To maintain the aspect ratio, letter and pillar boxing effects are used. The letter boxing operation adds black bars above and below the image. Pillar boxing is the vertical variant of letter boxing. As a result, the aspect ratio parameters of the image are maintained and non-symmetrical changes in the image size are compensated for using the black bars.

Table 9 shows examples of common scaling operations.

Table 8 *Scaling Operations*

Operation	VPP Input		VPP Output	
	Width x Height	CropX, CropY, CropW, CropH	Width x Height	CropX, CropY, CropW, CropH
Cropping	720x480	16,16,688,448	720x480	16,16,688,448
Resizing	720x480	0,0,720,480	1440x960	0,0,1440,960
Horizontal stretching	720x480	0,0,720,480	640x480	0,0,640,480
16:9 → 4:3 with letter boxing at the top and bottom	1920x1088	0,0,1920,1088	720x480	0,36,720,408
4:3 → 16:9 with pillar boxing at the left and right	720x480	0,0,720,480	1920x1088	144,0,1632,1088

4.10.4 Performing Frame Rate Conversion

Arbitrary frame rate conversion is supported by VPP. The application must specify the input and output frame rates using the FrameRateExtN and FrameRateExtD parameters in the mfxFrameInfo structure. The frame rate is defined as:

Frame rate = FrameRateExtN / FrameRateExtD.

Intel recommends using the frame rate provided by **DECODE** for the VPP input parameters.

If VPP is initialized to perform deinterlacing with an input frame rate of 29.97 and an output frame rate of 23.976, the Inverse Telecine algorithm is used.

Note that the frame rate conversion algorithm does not take into account input timestamps. Therefore, the number of output frames does not depend on any inconsistencies in the input timestamps. In the Intel Media SDK samples, the MFXVideoVPPEx class is demonstrates frame rate conversion based on timestamps.

See "<InstallFolder>\samples\sample_mfoundation_plugins\readme-mfoundation-plugins.rtf" for details.

More complex frame rate conversion approaches can be implemented as user plugins.

Please note that the Intel Media SDK implementation of frame rate conversion assumes constant frame rate. Do not use frame rate conversion for variable frame rate (VFR) inputs.

Many implementations of Intel® Iris™ Pro Graphics, Intel® Iris™ Graphics and Intel® HD Graphics (4200+ Series) allow support the creation of interpolated frame content by using the MFX_FRCALGM_FRAME_INTERPOLATION algorithm if the ratio of input-to-output frame rate is not supported, the VPP operation will report MFX_WRN_FILTER_SKIPPED. Commonly supported conversion ratios are 1:2 and 2:5 (useful for creating 60Hz content from 30Hz and 24Hz, respectfully).

4.11 Hint-based VPP filters

The second component of VPP includes optional video processing features such as denoising and procAmp, which are turned on and off with the help of hints. Hints are attached as external buffers to the mfxVideoParam structure and allow for enabling and disabling algorithms together with algorithm control parameters setting. The platform might not implement these operations, or it might not execute them in all situations*.

*Some platforms have global settings for properties such as Denoise. If this is enabled Media SDK denoise operation may be ignored.

VPP Filter	Config method	Description
Denoise	MFX_EXTBUFF_VPP_DENOISE	Remove image background noise. This is in addition to the deblocking filter inside the H.264 codec.
Detail enhancement	MFX_EXTBUFF_VPP_DETAIL	Enhance image details.
ProcAmp	MFX_EXTBUFF_VPP_PROCAMP	Correct image brightness, contrast, saturation and hue settings.
Image Stabilization	MFX_EXTBUFF_VPP_IMAGE_STABILIZATION	Reduces undesired frame-to-frame jitter commonly associated with the motion of a video camera

Figure 12 *Optional VPP Filters*

4.11.1 DoNotUse and DoUse lists

Some VPP functionality is turned on by default. If the application does not need a function, it can switch it off to increase performance. The pseudo-code example demonstrates how to configure the Intel® Media SDK VPP with a hint to disable denoising in the pipeline.

```
/* configure the DoNotUse hint */
mfxExtVPPDoNotUse dnu;
mfxU32 denoise=MFX_EXTBUFF_VPP_DENOISE;

dnu.Header.BufferId=MFX_EXTBUFF_VPP_DONOTUSE;
dnu.Header.BufferSz=sizeof(mfxExtVPPDoNotUse);
dnu.NumAlg=1;
dnu.AlgList=&denoise;

/* configure the mfxVideoParam structure */
mfxVideoParam conf;
mfxExtBuffer *eb=&dnu;

memset(&conf,0,sizeof(conf));
conf.IOPattern=MFX_IOPATTERN_IN_SYSTEM_MEMORY|
MFX_IOPATTERN_OUT_SYSTEM_MEMORY;
conf.NumExtParam=1;
conf.ExtParam=&eb;

conf.vpp.In.FourCC=MFX_FOURCC_YV12;
conf.vpp.Out.FourCC=MFX_FOURCC_NV12;
conf.vpp.In.Width=conf.vpp.Out.Width=1920;
conf.vpp.In.Height=conf.vpp.Out.Height=1088;

/* preprocessing initialization */
MFXVideoVPP_Init(session, &conf);
```

4.12 User Modules: Adding custom functionality to a pipeline

Intel® Media SDK 2.0 and later allows user-defined functions to participate in pipelines built from Intel Media SDK native functions. A user-defined module is essentially a set of functions that extend the Intel Media SDK functionality. Examples include a custom video preprocessing algorithm, or a building block that mixes/splits several video streams to build a complicated pipeline.

Although mixing user code and the Intel Media SDK functions in the application was possible before the 2.0 release, it required data synchronization between the user-defined functions and the Intel Media SDK functions. A key benefit of the newer API is that it allows integration of user-defined functions into an asynchronous Intel Media SDK pipeline.

For example, using a custom VPP plugin implemented as a user-defined module in a transcoding pipeline will not break the pipeline's asynchronous operation. This preserves the performance benefits of asynchronous pipeline.

4.12.1 Creating a new User Module

The API of the user-defined module is defined by the mfxPlugin structure. To create a new user-defined module, simply implement the following six call back functions: PluginInit, PluginClose, GetPluginParam, Submit, Execute, FreeResources. The function signatures are relatively transparent. (For function details, refer to the Intel Media SDK Reference Manual). Note that these functions are called by the Intel Media SDK itself, not by application.

The key functions here are Submit and Execute. They specify the user-defined module operation and provide integration into the Intel Media SDK task scheduling scheme.

Assume that the user-defined module performs Function(input, output) {algo}. Submit specifies the input and output argument structure of Function, which the Intel Media SDK scheduler treats as an abstract task. The SDK calls Submit to check the validity of the I/O parameters. If successful, Submit returns a task identifier to be queued for execution after the SDK resolves all input dependencies. Execute is the actual implementation of the Function algorithm. The SDK calls Execute for task execution after resolving all input dependencies.

4.12.2 Using a User-Defined Module in a Media Pipeline

After implementing a user-defined module, the next step is to integrate the module into the application based on the Intel Media SDK. To accomplish this, the Intel Media SDK 2.0 and later exposes the **USER** class of functions. The USER class is very similar to the **ENCODE**, **DECODE** and **VPP** class of functions, but with limited functionality.

Call the functions USER_Register, USER_Unregister and USER_ProcessFrameAsync from application code.

Note that user-defined functions are asynchronous. This means that the output is not ready when ProcessFrameAsync returns. The application must call SyncOperation at the corresponding sync_point to get the function output when it is available. Alternatively, the output can be pipelined to other SDK functions without synchronization. (See the Intel Media SDK Reference Manual for details on asynchronous operation.)

A pseudo-code example of how to integrate a user-defined module into a transcoding pipeline is shown below.

```
MFXVideoUSER_Register(session,0,&my_user_module);
MFXVideoDECODE_Init(session, decoding_configuration);
MFXVideoVPP_Init(session, preprocessing_configuration);
/* Initialize my user module */
MFXVideoENCODE_Init(session, encoding_configuration);
do {
   /* load bitstream to bs_d */
   MFXVideoDECODE_DecodeFrameAsync(session, bs_d, surface_w, &surface_d, &sync_d);
   MFXVideoVPP_RunFrameVPPAsync(session, surface_d, surface_v, NULL, &sync_v);
   MFXVideoUSER_ProcessFrameAsync(session, &surface_v, 1, &surface_u, 1, &sync_u);
   MFXVideoENCODE_EncodeFrameAsync(session, NULL, surface_u, bs_e, &sync_e);
   MFXVideoCORE_SyncOperation(session, sync_e, INFINITE);
/* write bs_e to file */
} while (!end_of_stream);

MFXVideoENCODE_Close(session);
/* Close my user module */
MFXVideoVPP_Close(session);
MFXVideoDECODE_Close(session);
MFXVideoUSER_Unregister(session);
```

4.12.3 Implementation Example

Consider a simple example of a user-defined module implementation—a 180-degree frame rotation. The code segments below were adapted from the sample code included with the Intel® Media SDK. (Refer to the files sample_plugin.cpp and pipeline_user.cpp located in "<InstallFolder>\samples\sample_encode\src\" for a fully functional implementation.)

A rotation module task is associated with a single frame and is bundled with input and output frame surfaces. The 180-degree rotation algorithm requires two swapping operations. First, pixels within each line are swapped with respect to the middle pixel. Then lines are swapped with respect to the middle line of the image. The algorithm can be parallelized to process image lines in chunks, since lines can be processed independently. Call Execute several times for a particular task (frame) until the frame is fully processed. The code for this example is shown below.

```
// Submit function implementation
mfxStatus Rotate::mfxSubmit(const mfxHDL *in, mfxU32 in_num, const mfxHDL *out,
mfxU32 out_num, mfxThreadTask *task)
{
    ...
    mfxFrameSurface1 *surface_in = (mfxFrameSurface1 *)in[0];
    mfxFrameSurface1 *surface_out = (mfxFrameSurface1 *)out[0];
    mfxStatus sts = MFX_ERR_NONE;
   // check validity of parameters
    sts = CheckInOutFrameInfo(&surface_in->Info, &surface_out->Info);
    CHECK_RESULT(sts, MFX_ERR_NONE, sts);
    mfxU32 ind = FindFreeTaskIdx();

    if (ind >= m_Param.MaxNumTasks)
    {
        // currently there are no free tasks available
        return MFX_WRN_DEVICE_BUSY;
    }
    m_pTasks[ind].In = surface_in;
    m_pTasks[ind].Out = surface_out;
    m_pTasks[ind].bBusy = true;
    switch (m_Param.Angle)
    {
    case 180:
        m_pTasks[ind].pProcessor = new Rotator180;
        CHECK_POINTER(m_pTasks[ind].pProcessor, MFX_ERR_MEMORY_ALLOC);
        break;
    default:
        return MFX_ERR_UNSUPPORTED;
    }

    m_pTasks[ind].pProcessor->Init(surface_in, surface_out);

    *task = (mfxThreadTask)&m_pTasks[ind];

    return MFX_ERR_NONE;
}
```

```
// Execute function n implementation
mfxStatus Rotate::mfxExecute(
                        mfxThreadTask task, mfxU32 thread_id, mfxU32 call_count)
{
    mfxStatus sts = MFX_ERR_NONE;
    RotateTask *current_task = (RotateTask *)task;
    // 0,...,NumChunks - 2 calls return TASK_WORKING, NumChunks - 1,.. return
TASK_DONE
    if (call_count < m_NumChunks)
    {
        // there's data to process
        sts = current_task->pProcessor->Process(&m_pChunks[call_count]);
        CHECK_RESULT(sts, MFX_ERR_NONE, sts);
        // last call?
        sts = ((m_NumChunks - 1) == call_count)?MFX_TASK_DONE : MFX_TASK_WORKING;
    }
    else
    {
        // no data to process
        sts =  MFX_TASK_DONE;
    }
    return sts;
}
```

4.13 Utilizing 3D functionality and content

While the Intel® Media SDK provides support for MVC stereo streams, there is also a desire to display the streams. The ability to display unique images intended for the left and right eye requires the involvement of display hardware, and there have been many solutions used in various industries to achieve this.

To display Stereo 3D content on platforms that support Microsoft* Direct3D 11.1, please see Microsoft documentation and sample code found here:
http://go.microsoft.com/fwlink/p/?linkid=238402

For Microsoft* Windows* 7 platforms using 2nd Generation Intel® Core™ processors or later, a static library (.lib) is provided as part of the Intel Media SDK that allows creation of a `GFXS3DControl` object that can be used for detection and control of stereo 3D-capable televisions and embedded DisplayPort* devices. These display devices (monitors) may offer a wide variety of PC-to-display configuration options, and the options offered can vary.

Information about this control library can be found here: http://software.intel.com/file/38355.

Some stereo 3D televisions allow the user to manually select the method it uses to interpret the data being received. Televisions that use HDMI* 1.4 allow this selection to be made programmatically. Televisions supporting this standard are not required to support all possible methods of receiving stereo images from a device attached to it. The PC and display device must negotiate a common, compatible method of outputting display data, and this is a necessary first step when displaying 3D content. The `GFXS3DControl` object provides

information about which modes are available for stereoscopic display. Once this is known the mode can be enabled, and all desktop content will be sent as identical left and right images, except for the cursor, which will only appear on the left view.

Some stereoscopic displays require activation of "active glasses" or other manual tasks to correctly display the 3D content they are receiving, and there is nothing the `GFXS3DControl` control can do to automate this display-specific part of the process.

Querying and setting a stereoscopic 3D display mode

```
#include "igfx_s3dcontrol.h"
…
IGFX_S3DCAPS   s3DCaps = {0};  // platforms S3D capabilities
IGFX_DISPLAY_MODE mode = {0};  // platform display mode

// Create GFXS3DControl opbject
IGFXS3DControl* m_pS3DControl = CreateIGFXS3DControl();

// Get  platform S3D capabilities
m_pS3DControl->GetS3DCaps(&s3DCaps);

// get list of supported S3D display modes
ULONG pref_idx = 0xFFFF;
for (ULONG i = 0; i<s3DCaps.ulNumEntries; i++)
{
    if (Less(s3DCaps.S3DSupportedModes[pref_idx], s3DCaps.S3DSupportedModes[i]))
       pref_idx = i;
}
If (0xFFFF == pref_idx)
{
    // no S3D mode found
    return MFX_ERR_UNSUPPORTED;
}
mode = m_Caps.S3DSupportedModes[pref_idx];

// and finally set the mode.  It may take some time for some monitors to actually
start display the mode.
m_pS3DControl->SwitchTo3D(&mode);
```

Once a Stereo 3D mode is enabled, two DXVA2 video processors can be created, one for the left channel and one for the right. This is accomplished by using a method available in the DXVA2VideoService to select the view prior to creating each Video Processor object.

To render different content to left and right views, a DXVA2 `VideoProcessBlt` call is needed for each video processor. Once both views have been rendered, a single call to Direct3D* Present call will cause the display controller to continuously send both the left and right views to the monitor. All content that is not part of the render output of these DXVA2 processors (for example, the Windows* desktop) will be displayed to both the left and right views. Two `VideoProcessBlt` operations should occur between each Present call, to update left and right views.

When all stereoscopic displaying is complete, an application can return the system to 2D display mode by calling the `SwitchTo2D` method of the `GFXS3DControl`.

Please see the "sample_decode" sample code and readme for detailed examples of the use of the `IGFXS3DControl` static library and the `IGFXS3DControl interface.`

4.14 Inserting SEI message into encoded AVC stream

Intel® Media SDK supports insertion of any SEI message into the encoded stream. This is achieved by adding payload (mfxPayload) to the encoded frame by specifying the encoder parameters via mfxEncodeCtrl.

The following code example showcases how to insert the SEI message type "user_data_registered_itu_t_t35", commonly used to carry closed captioning information.

```
#define SEI_USER_DATA_REGISTERED_ITU_T_T35  4

typedef struct
{
  unsigned char countryCode;
  unsigned char countryCodeExtension;
  unsigned char payloadBytes[10];  // Containing arbitrary captions
} userdata_reg_t35;

// Insert SEI_USER_DATA_REGISTERED_ITU_T_T35 into payload
userdata_reg_t35 m_userSEIData;
m_userSEIData.countryCode           = 0xB5;
m_userSEIData.countryCodeExtension  = 0x31;
m_userSEIData.payloadBytes[0]       = 0x41; // payloadBytes[] containing captions

mfxU8 m_seiData[100]; // Arbitrary size
mfxPayload m_mySEIPayload;
MSDK_ZERO_MEMORY(m_mySEIPayload);
m_mySEIPayload.Type    = SEI_USER_DATA_REGISTERED_ITU_T_T35;
m_mySEIPayload.BufSize = sizeof(userdata_reg_t35) + 2;  // 2 bytes for header
m_mySEIPayload.NumBit  = m_mySEIPayload.BufSize * 8;
m_mySEIPayload.Data    = m_seiData;

// Insert SEI header and SEI msg into data buffer
m_seiData[0] = (mfxU8)m_mySEIPayload.Type;         // SEI type
m_seiData[1] = (mfxU8)(m_mySEIPayload.BufSize – 2) // Size of following msg
memcpy(m_seiData+2, &m_userSEIData, sizeof(userdata_reg_t35));

mfxPayload* m_payloads[1];
m_payloads[0] = &m_mySEIPayload;

// Encode control structure initialization
mfxEncodeCtrl m_encodeCtrl;
MSDK_ZERO_MEMORY(m_encodeCtrl);
m_encodeCtrl.Payload = (mfxPayload**)&m_payloads[0];
m_encodeCtrl.NumPayload = 1;

// Use encode control structure while calling encode
m_pmfxENC->EncodeFrameAsync(&m_pEncodeCtrl,
                            &m_pEncSurfaces[nEncSurfIdx],
                            &pCurrentTask->mfxBS,
                            &pCurrentTask->EncSyncP);
```

4.15 Custom control of encoded AVC SPS/PPS data

Intel® Media SDK provides direct control of a subset of SPS/PPS header parameters defined by the AVC standard. For control of specific SPS/PPS parameters that cannot be controlled via the SDK API, please use the following method:

1) Configure encoder with a set of parameters closest matching the desired SPS/PPS set
2) After initalizing the encoder call GetVideoParam() with extended buffer mfxExtCodingOptionSPSPPS to extract the SPS and PPS selected by the encoder (make sure to allocate sufficient space for storage of SPS and PPS buffer)
3) Make the desired modifications to SPS and/or PPS buffer
4) Apply the changes to the encoder by calling Reset(), referencing the new mfxVideoParam structure including the mfxExtCodingOptionSPSPPS extended buffer. Note that, when using this method the SPS/PPS parameters set via mfxVideoParams will be overwritten by the custom SPS/PPS buffer

The simple example below shows how to manually control SPS "constraint_set" values. In this case we are setting "constraint_set1" flag to 1 (indicates constrained baseline profile). The same approach can be used to control any SPS or PPS parameters manually.

NOTE: The Media SDK API was extended as part of Media SDK 2012 R2 release. This API, 1.4, now allows direct API control to set Constrained Baseline Profile (MFX_PROFILE_AVC_CONSTRAINED_BASELINE). Manual SPS control is not required.

```
// ... Encoder initialized

mfxExtCodingOptionSPSPPS m_extSPSPPS;
MSDK_ZERO_MEMORY(m_extSPSPPS);
m_extSPSPPS.Header.BufferId = MFX_EXTBUFF_CODING_OPTION_SPSPPS;
m_extSPSPPS.Header.BufferSz = sizeof(mfxExtCodingOptionSPSPPS);

// Allocate sufficient space for SPS/PPS buffers (100 is enough)
m_extSPSPPS.PPSBuffer = (mfxU8*)malloc(100);
m_extSPSPPS.SPSBuffer = (mfxU8*)malloc(100);
m_extSPSPPS.PPSBufSize = 100;
m_extSPSPPS.SPSBufSize = 100;

// Add SPSPPS extended buffer to encoder parameter extended buffer list
//    std::vector<mfxExtBuffer*> m_EncExtParams
//    mfxVideoParam m_mfxEncParams
m_EncExtParams.push_back((mfxExtBuffer *)&m_extSPSPPS);
m_mfxEncParams.ExtParam = &m_EncExtParams[0];
m_mfxEncParams.NumExtParam = (mfxU16)m_EncExtParams.size();

// Retrieve selected encoder parameters
mfxStatus sts = m_pmfxENC->GetVideoParam(&m_mfxEncParams);

// Very simple representation of first set of SPS buffer members
typedef struct {
       unsigned char NAL_bytes[5];
       unsigned char SPS_Profile;
       unsigned char SPS_Constraint_Set;
       unsigned char SPS_Level;
       // ... rest of SPS is ignored (needs parsing)
} BasicSPS;
BasicSPS* mySPS = (BasicSPS*) m_extSPSPPS.SPSBuffer;
mySPS->SPS_Constraint_Set |= 0x40; // Set constraint_set1 to 1

// Reset encoder with modified SPS buffer
sts = m_pmfxENC->Reset(&m_mfxEncParams);
```

4.16 Handling concurrent Media SDK pipelines

Intel® Media SDK does not limit the number of concurrent sessions (regardless of the components used in each session). Effectively this means that an application can run several independent Media SDK sessions simultaneously. Individual session throughput will naturally be lower, compared to single session throughput since the sessions will share the resources. Please note however that multisession performance gains are not linear.

An easy way to achieve concurrency is to host each full session in separate thread as illustrated in the example below (based on Media SDK sample_decode sample).

```
long    threadCount      = 0;
HANDLE  ThreadsDoneEvent = NULL;
// Using InterlockedDecrement/Event to avoid 64 limit of WaitForMultipleObjects
void DecrementThreadCount() {
  long value = InterlockedDecrement(&threadCount);
  if(value == 0) SetEvent(ThreadsDoneEvent);
}
void IncrementThreadCount() {
  long value = InterlockedIncrement(&threadCount);
  if(value > 0) ResetEvent(ThreadsDoneEvent);
}

DWORD WINAPI DecodeThread(LPVOID arg)
{
  IncrementThreadCount();

  mfxStatus sts = MFX_ERR_NONE;
  sInputParams *pParams = (sInputParams *)arg;
  CDecodingPipeline Pipeline;

  sts = Pipeline.Init(pParams);
  MSDK_CHECK_RESULT(sts, MFX_ERR_NONE, 1);

  sts = Pipeline.RunDecoding();
  MSDK_CHECK_RESULT(sts, MFX_ERR_NONE, 1);
  // For simplicity, error handling (such as in original decode sample) ignored

  Pipeline.Close();

  DecrementThreadCount();

  return 0;
}

int _tmain(int argc, TCHAR *argv[])
{
  ThreadsDoneEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
  std::vector<sInputParams> paramVector;
  // … initialize parameter struct for each file to decode
  int numMedia = paramVector.size();
  HANDLE* pDecodeThreads = new HANDLE[numMedia];

  for(int i=0; i<numMedia; ++i)
    pDecodeThreads[i] =
       CreateThread(NULL, 0, DecodeThread, (LPVOID)&paramVector[i], 0, NULL);

  WaitForSingleObject(ThreadsDoneEvent, INFINITE);

  for(int i=0; i<numMedia; ++i)
    CloseHandle(pDecodeThreads[i]);

  delete [] pDecodeThreads;
  CloseHandle(ThreadsDoneEvent);
}
```

There are some things to keep in mind when executing several Media SDK sessions concurrently:

- If processing large number of streams with high resolution on a 32bit OS you may run into memory limitations. The error message provided by Media SDK when reaching memory limit is not very descriptive (due to underlying vague return codes from OS)
- If you encounter this limitation, please consider using 64 bit OS (greater pool of available graphics memory)

4.17 Using Media SDK to build Video conferencing or streaming applications

Intel® Media SDK 2012 (API 1.3) extended support for video conferencing and streaming. The features listed below addresses common video conferencing and streaming requirements for improved adaptation to transmission conditions, robustness and real-time responsiveness:

- Low Latency Encode and Decode
- Dynamic Bit Rate Control
- Dynamic Resolution Control
- Forced Key Frame Generation
- Reference List Selection
- Reference Picture Marking Repetition SEI message
- Rolling I-Frame support (*1.6 API*)
- Long Term Reference Frame (LTR)
- Temporal Scalability
- Motion JPEG (MJPEG) Decode

Please refer to the paper, http://software.intel.com/en-us/articles/video-conferencing-features-of-intel-media-software-development-kit/, for details about using Media SDK in the video conferencing and streaming context.

Note: The white paper describes how to configure encoder for temporal scalability but not how the decoder application must interpret the encoded stream.
For temporal streams "nal_unit_header_svc_extension" is attached to each slice header, as described in standard (G.7.3.1.1). The headers contains "temporal_id", which can be used to extract a certain temporal layer. Logic is simple: skip all slices with temporal_id > target_temporal_id and keep all slices with temporal_id <= target_temporal_id.

4.18 Handling Multiple Graphics Adapters

Some platforms that support Intel hardware acceleration may also include discrete graphics adapters. Desktop platforms may have 3rd party graphics adapters added by OEMs or end-users. Many OEM laptop system 's also include a discrete card , and allow the end user (or even applications) to dynamically choose which adapter is active. This is often referred to as "switchable graphics". The increasing usage models can quickly lead to confusion when planning your application.

Prior to the introduction of the Intel Media SDK 2013, supporting hardware acceleration in a multi-gpu environment was bound by a fundamental constraint - the Intel Graphics adapter needed to have a monitor associated with the device to be active. This constraint was due to the capabilities of the Microsoft DirectX 9 infrastructure that the Intel Media SDK, and associated graphics driver were based upon. The introduction and corresponding support of the DirectX 11 in both the Intel Media SDK and graphics driver has now simplified the process of developing applications to utilize Intel's fixed function hardware acceleration, even when the Intel graphics device is not connected to an external display device.

4.18.1 Muti-GPU and "headless" configurations

Applications wishing to leverage the Intel Media SDK's hardware acceleration library when a discrete card is the primary device, or on devices without a monitor attached – such as "Session 0" modes, are required to initialize the Intel Media SDK to utilize the DirectX11 infrastructure, as well as provide its own memory allocation routines that manage DirectX 11 surfaces.

The following code illustrates the correct initialization parameters for initializing the library. MFX_IMPL_AUTO_ANY will scan the system for a supporting adapter, while MFX_IMPL_VIA_D3D11 indicates the need for DirectX 11 support.

```
// Initialize Media SDK session
// - MFX_IMPL_AUTO_ANY selects HW accelaration if available (on any adapter)

mfxIMPL impl = MFX_IMPL_AUTO_ANY | MFX_IMPL_VIA_D3D11;
mfxVersion ver = {0, 1};
MFXVideoSession mfxSession;

sts = mfxSession.Init(impl, &ver);
MSDK_CHECK_RESULT(sts, MFX_ERR_NONE, sts);
```

In addition to initializing the library, the application also needs to create the correct DirectX device context on the correct adapter. The following code is a simplistic illustration of how to enumerate the available graphics drivers on the system, and create the DirectX device on appropriate adapter. In this case the `g_hAdapter` handle is actually pointing to the Intel adapter which is in the secondary position.

```
mfxStatus CreateHWDevice(mfxSession session, mfxHDL* deviceHandle)
{
    HRESULT hres = S_OK;

    static D3D_FEATURE_LEVEL FeatureLevels[] = {
        D3D_FEATURE_LEVEL_11_1,
        D3D_FEATURE_LEVEL_11_0,
        D3D_FEATURE_LEVEL_10_1,
        D3D_FEATURE_LEVEL_10_0
    };
    D3D_FEATURE_LEVEL pFeatureLevelsOut;

       g_hAdapter = GetIntelDeviceAdapterHandle(session);
       if(NULL == g_hAdapter)
              return MFX_ERR_DEVICE_FAILED;

    hres =  D3D11CreateDevice(g_hAdapter,
                              D3D_DRIVER_TYPE_UNKNOWN,
                              NULL,
                              0,
                              FeatureLevels,
                              (sizeof(FeatureLevels) /sizeof(FeatureLevels[0])),
                              D3D11_SDK_VERSION,
                              &g_pD3D11Device,
                              &pFeatureLevelsOut,
                              &g_pD3D11Ctx);
    if (FAILED(hres))
        return MFX_ERR_DEVICE_FAILED;

       g_pDXGIDev           = g_pD3D11Device;
       g_pDX11VideoDevice   = g_pD3D11Device;
       g_pVideoContext      = g_pD3D11Ctx;


       // turn on multithreading for the Context
    CComQIPtr<ID3D10Multithread> p_mt(g_pVideoContext);
    if (p_mt)
        p_mt->SetMultithreadProtected(true);
    else
        return MFX_ERR_DEVICE_FAILED;

       *deviceHandle = (mfxHDL)g_pD3D11Device;

    return MFX_ERR_NONE;
}
```

```
IDXGIAdapter* GetIntelDeviceAdapterHandle(mfxSession session)
{
    mfxU32    adapterNum = 0;
    mfxIMPL impl;

    MFXQueryIMPL(session, &impl);

    // Extract Media SDK base implementation type
    mfxIMPL baseImpl = MFX_IMPL_BASETYPE(impl);

   // get corresponding adapter number
    for (mfxU8 i = 0; i < sizeof(implTypes)/sizeof(implTypes[0]); i++)
    {
        if (implTypes[i].impl == baseImpl)
        {
            adapterNum = implTypes[i].adapterID;
            break;
        }
    }

    HRESULT hres =
CreateDXGIFactory(__uuidof(IDXGIFactory2),(void**)(&g_pDXGIFactory) );
    if (FAILED(hres)) return NULL;

    IDXGIAdapter* adapter;
    hres = g_pDXGIFactory->EnumAdapters(adapterNum, &adapter);
    if (FAILED(hres)) return NULL;

       return adapter;
}
```

Finally, applications also need to ensure they are using the appropriate DirectX 11 routines for its memory allocator. Examples for this are available as part of the Media SDK sample code.

4.18.2 Mutliple-Monitor configurations

When a system supports display output to multiple monitors, each display is connected to the output of a graphics adapter. One graphics adapter may support multiple displays. For example, on many systems Intel's processor graphics adapter may be connected to an analog VGA* monitor and to an HDMI* monitor. The Windows* operating system will assign the primary (or "main") display as the default display device and use the associated adapter as the acceleration device if the application does not provide specific information about which device it wants to use. If the system contains one or more discrete graphics adapters in addition to the integrated graphics adapter, the user can set a display connected to the discrete adapter to be the primary display. In this case, applications that request the services of the default display adapter will not be able to take advantage of Intel's Quick Sync Video acceleration. When an application creates an Media SDK session using the 'default' adapter, a call to MFXQueryIMPL() may not return a MFX_IMPL_HARDWARE status if the default adapter does not provide acceleration support for the MediaSDK API.

It is recommended that applications desiring hardware acceleration consider initializing their Media SDK session with MFX_IMPL_AUTO_ANY or MFX_IMPL_HARDWARE_ANY to allow all the capabilities of all the available graphics adapters to be examined, not just the 'default' adapter. When either of these options are used, calls to MFXQueryIMPL() will return information about which device contains hardware acceleration. The application can use response of MFX_IMPL_HARDWARE2, MFX_IMPL_HARDWARE3 and MFX_IMPL_HARDWARE4 to know which device should be used. The application should ensure that the creation of any Microsoft* Direct3D* device use the appropriate adapter, and not the default adapter, as the resources associated with the default adapter may not support acceleration.

4.18.3 Switchable Graphics configurations

Some platforms allow the user to dynamically select which graphics adapter is responsible for rendering to a single display for specific applications. For example, a user can select a game to execute on a discrete graphics adapter and a media application to execute on Intel's integrated graphics adapter. On these "switchable graphics" systems, the primary display adapter is changed to match the desired adapter for each application (process). The resources and capabilities of the inactive adapter are not available to applications, as the only adapter seen by the application is the current, active adapter. From the application's point of view, the Media SDK operations may report MFX_WRN_PARTIAL_ACCELERATION if a session is initialized for hardware acceleration but the currently configuration does not allow acceleration to be used.

5 Troubleshooting Intel® Media SDK

5.1 Debugging toolchest

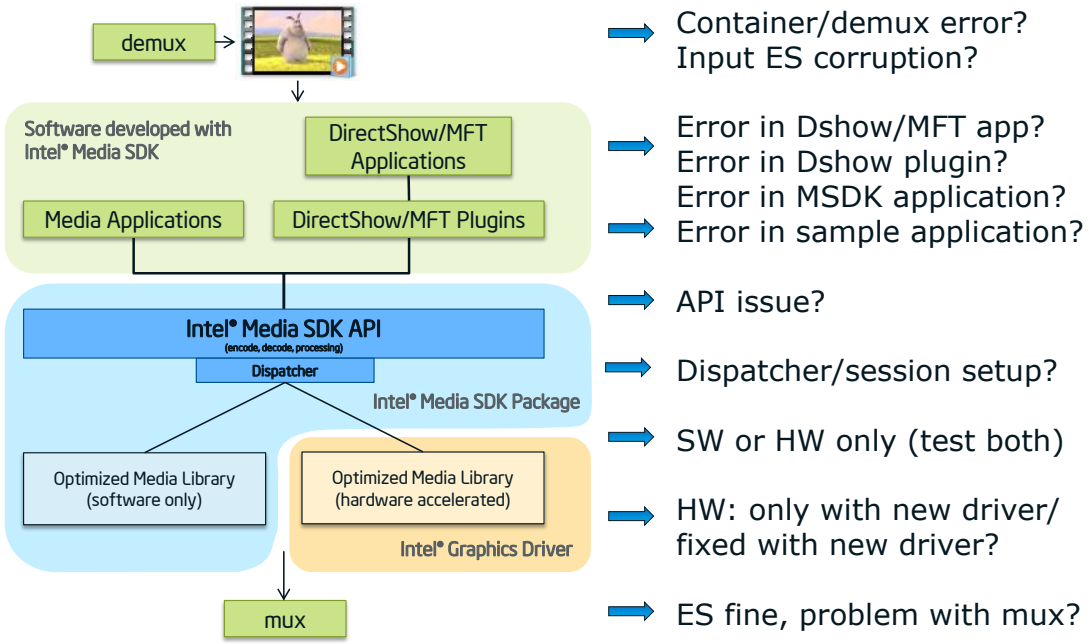
Intel® Tools/SDKs

- Intel® Media SDK Tracer
- Intel® Media SDK System Analyzer
- Intel® Graphics Performance Analyzers (Intel® GPA)

Useful tools

- TightVNC* (remote access + hardware acceleration)
- Elecard* tools (stream and codec-level analysis)
- MediaInfo (media meta data details)
- ffmpeg (versatile media processing tool)
- ffprobe (stream/packet details, shows parsing/nonconformant stream errors)
- Process explorer (memory leaks, thread inspection, etc.)

5.2 Root cause by architecture

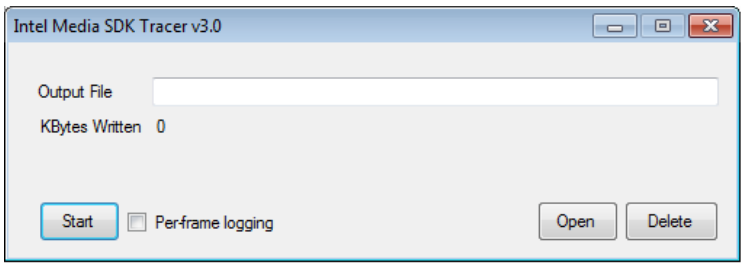


Suspected problem layer	Tests
Container/demux	Timestamps in orig, use different demuxer

Input ES corruption	Does input stream work with other decoders (ffmpeg, reference decoder, etc.)
App built using DirectShow*/MFT plugin	Problem also in simplified GraphEdit* graph?
DirectShow/MFT plugin	Problem still there with similar filter (i.e. different MPEG-2 decode)
Intel® Media SDK-based application	**Intel® GPA trace**: which components is app using? **MediaSDK_tracer tool**: Which parameters are used at decode/VPP/encode setup? For each frame? Set up a similar scenario with a sample. Problem reproduced?
Sample application	Can the problem be localized to a specific stage of the sample app? Any warnings/error messages?
Intel® Media SDK API	Isolate stages with intermediate YUV IO
Dispatcher/session	MediaSDK_tracer (dispatcher logs) Process explorer (memory leaks, thread pool, session startup)
Software only	Does problem also exist in hardware? Software versions of different releases?
Hardware only	Also in software? Outputs change with different graphics driver versions? Different platforms?

5.3 MediaSDK_tracer tool

The MediaSDK_tracer tool is available as a separately downloadable component via the Media SDK portal web page. It can be used to log Intel Media SDK activity to a file. Since this includes all parameters for Intel Media SDK sessions and pipeline stages, potentially for each frame, the output log for even a short run can be several megabytes. However, much of the power of this utility comes from its exhaustiveness – it captures a text representation of Intel Media SDK state which can be analyzed and compared to other runs. For more information, see the MediaSDK_tracer readme.



To use:

1. Specify an output file. By default, logging for encode, decode, VPP, and sync is disabled. To enable, check per-frame logging. Note: the tracer tool appends to the log file.
2. Press Start.
3. The tool can capture from a running process, but it is often helpful to start the application after starting the trace tool.

5.4 MediaSDK_sys_analyzer tool

The Media SDK "sys_analyzer" tool is available as a separately downloadable component via the Media SDK portal web page. The utility is used to analyze the developer platform, reporting back Media SDK related capabilities. The tool may help find environment, system configuration or driver installation issues that may prevent taking advantage of the full capabilities of the Intel Media SDK. For more information please refer to the documentation associated with the tool.

5.5 Intel® Graphics Performance Analyzers (Intel® GPA)

The Intel® Graphics Performance Analyzer is a free tool available via Intel® Visual Computing Source website: http://software.intel.com/en-us/vcsource/tools/intel-gpa/

The tool provide an easy-to-use suite of optimization tools for analyzing and optimizing games, media, and other graphics intensive applications.

Please also refer to the following paper for details on how to analyze Media SDK application using the Intel® GPA tool.
http://software.intel.com/en-us/articles/using-intel-graphics-performance-analyzer-gpa-to-analyze-intel-media-software-development-kit-enabled-applications/

6 Appendixes

6.1 Encoder Configuration for Blu-ray* and AVCHD*

This section describes how to configure the Intel® Media SDK library to encode H.264 and MPEG-2 video data in compliance with Blu-ray* or AVCHD* formats. For details on those formats, consult the following documents:

- AVCHD Format, Version 2.0, Book 1: Playback System Basic Specifications
- System Description Blu-ray Disc Read-Only Format, Part 3, Audio Visual Basic Specifications, Version 2.41, August 2010

The user may also need to consult other documents about Blu-ray and AVCHD. Additionally, be aware that you may be required to have licenses from other companies (not Intel Corporation) to access Blu-ray or AVCHD specifications. This section provides information about the Intel Media SDK library and its data structures to help you comply with Blu-ray or AVCHD specifications.

6.1.1 Encoder Configuration

Configure SDK library encoders in the customary manner:

1. Specify the Profile, Level, Width, Height, AspectRatio, ChromaFormat and other parameters in the mfxVideoParam structure;
2. Set the parameter MaxDecFrameBuffering or others in the mfxExtCodingOption structure to improve encoder results.
3. Explicitly specify the NumSlice value.

6.1.2 GOP Sequence

SDK encoders use a MPEG-2 style of GOP sequence control for both MPEG-2 and H.264 formats. Generate the desired GOP sequence by configuring:

- GopPicSize, GopRefDist, GopOptFlag and IdrInterval in the mfxVideoParam structure. The MFX_GOP_STRICT flag must be specified.
- ResetRefList in the mfxExtCodingOption structure.

Applications can collect information about a desired GOP sequence and manually insert a GOP structure map into the bitstream with the following procedure:

1. Create a “fake” or dummy GOP structure map in accordance with the Blu-ray* specification;
2. Before a GOP sequence begins, insert the “fake” GOP structure map via payload insertion through the mfxEncodeCtrl structure;

3. Encode the GOP sequence. After encoding each frame in the sequence, the PicStruct member of the mfxBitstream structure reports the frame's picture type;
4. Replace the "fake" payload with the real, application-constructed GOP structure map.

6.1.3 SPS and PPS

SDK encoders write a single set of SPS and PPS NAL units to the output bitstream. Use EndofSequence, EndofStream, PicTimingSEI and AUDelimiter in the mfxExtCodingOption structure to configure how encoders write NAL units or SEI messages to the bitstream.

6.1.4 HRD Parameters

The Intel® Media SDK encoder supports HRD Type II compliance, but only with a single set of HRD parameters:

- Configurable parameters in the SDK related to HRD include RateControlMethod, InitialDelayInKB, BufferSizeInKB, TargetKbps, and MaxKbps in the mfxVideoParam structure.
- Additionally, the user can specify VuiNalHrdParameters in the mfxExtCodingOption structure to configure how the application writes HRD parameters to the bitstream.

6.1.5 Preprocessing Information

If the input pictures are preprocessed using 3:2 pull down, frame doubling or tripling, the application must convey this information through the PicStruct member of the **mfxFrameSurface1** structure during encoding.

6.1.6 Closed-Captioned SEI Messages

Applications can construct their own closed-captioned SEI messages (for H.264), or picture user data (for MPEG-2 video). Implement payload insertion through the mfxEncodeCtrl structure to put them into the output bitstream.

Please refer to "Inserting SEI message into encoded AVC stream" chapter for details on how to insert custom SEI message into H.264 streams. Can be used for closed captioning.

6.1.7 Unsupported Features

The Intel® Media SDK does not support:

- Decoder reference picture marking syntax; in this case, Blu-ray* requirements do not apply.
- Color matrix parameters, such as colour_primaries, transfer_characteristics or matrix_coefficients from the H.264 specification; the SDK does not write these into the bitstream.
- All NAL unit types or SEI messages; however, the SDK supports those needed by Blu-ray or AVCHD* and that are also defined in the H.264 specification.

6.1.8 Additional configuration for H.264 Stereo High Profile

Intel® Media SDK 2012 R2 introduces a new 3D video (MVC) encoding mode as part of API 1.4. This mode is enabled by the ViewOutput flag in mfxExtCodingOption structure; it instructs H.264 Encoder to output views in separate bitstream buffers and meet a set of bitstream packing conditions. Most significant of those conditions are listed below:

- Slice NAL units for each view are preceded by corresponding headers for each view (SPS, PPS, SEI).
- Prefix NAL unit with ID 0x14 is not inserted before the base view slice NAL units.
- View dependency representation delimiter (VDRD) NAL unit with ID 0x18 is inserted before header NAL units for the dependent view.
- If Buffering period and/or Picture timing SEI message(s) are present in base view component then corresponding dependent view component contains MVC scalable nested SEI with such SEI message(s). Dependent view conforms to HRD conformance with hrd_parameters() encoded in SPS/VUI inside Subset SPS and with mentioned BP, PT SEI messages encapsulated into MVC nested SEI

Applications should turn this mode on if they intend to produce H.264 Stereo High Profile data in compliance with Blu-ray* or AVCHD* formats.

6.2 Encoder Configuration for DVD-Video*

The Intel® Media SDK MPEG-2 encoder allows for producing DVD-Video*-compatible video streams. This section describes how to configure the Intel Media SDK library to encode MPEG-2 video data in compliance with DVD-Video format. For details on the format itself, consult the following documents:

- DVD Read Only Specifications Book, Part 3, DVD-Video Book (Book B)

The user may also need to consult other documents about DVD-Video. Additionally, be aware that you may be required to have licenses from other companies (not Intel Corporation) to access DVD-Video specifications. This section provides information about the Intel Media SDK library and its data structures to help you comply with DVD-Video specifications.

6.2.1 Encoder Configuration

Configure MPEG-2 video encoder in the customary manner:

Specify the Profile, Level, Width, Height, AspectRatio, ChromaFormat and other parameters in the mfxVideoParam structure. Note that the Intel® Media SDK accepts Pixel Aspect Ratio while applications likely operate with Display Aspect Ratio. An accurate conversion is needed.

For example for DAR = 4:3 PAR will be calculated as follows:

FrameInfo.AspectRatioW = 2304(4 * 576)
FrameInfo.AspectRatioH = 2160(3 * 720)

6.2.2 GOP Sequence

Specify the desired GOP sequence by configuring:

GopPicSize, GopRefDist, GopOptFlag and IdrInterval in the mfxVideoParam structure.

6.2.3 Setting NTSC/PAL video_format in the sequence_display_extension header

NTSC and PAL are analog encoding systems for broadcast television. Sometimes, applications require streams produced to specify the encoding system used. The Intel® Media SDK does not contain any NTSC or PAL encoding functionality. However, the encoder can write flags into the bitstream specifying one format or the other. The application is responsible for setting the encoding parameters consistent to the respective specification.

The data that identifies PAL or NTSC is stored in the video_format value of the sequence_display_extension header. Since this data is purely decorative to the Intel Media SDK, it is not directly accessible through the mfxVideoParam structure. Instead, the application can make the Encoder write this data into the stream by specifying a mfxExtCodingOptionSPSPPS structure with the required headers. This method allows the application to define custom SPS and PPS information in the stream.

Please refer to "Custom control of encoded AVC SPS/PPS data" chapter for details on manual customizations to SPS/PPS data.

6.2.4 Preprocessing Information

If the input pictures are preprocessed—3:2 pull down, frame doubling or tripling—the application should convey such information through the PicStruct member of the mfxFrameSurface1 structure during encoding.

6.2.5 Closed Captioning

Applications can construct their own closed-captioned picture user data for MPEG-2 video. Implement payload insertion through the mfxEncodeCtrl structure to put them into the output bitstream.