# Decoding Sample

## Overview

**Decoding Sample** works with **Intel® Media Server Studio 2018 - SDK for Linux* Server** (hereinafter referred to as "**SDK**").

It demonstrates how to use the **SDK** API to create a sample console application that performs decoding of various video compression formats.

The sample can work together with **Intel® Media Server Studio – HEVC Decoder & Encoder** (hereinafter referred to as "**HEVC**").

**Note:** To run HEVC, please read the instructions in the “HEVC Plugin” section carefully.

## Features

**Decoding Sample** supports the following video formats:

| **Format type** | |
|---|---|
| input (compressed) | H.264 (AVC, MVC – Multi-View Coding), MPEG-2 video, VC-1, JPEG*/Motion JPEG, HEVC (High Efficiency Video Coding), VP8 |
| output (uncompressed) | YUV420 |

**Note 2: Decoding Sample** renders the decoded video stream to a file in YUV 4:2:0 sampling format, with the color planes Y, U and V in that order.

## Hardware Requirements

See `<install-folder>\Media_Samples_Guide.pdf.`

## Software Requirements

See `<install-folder>\Media_Samples_Guide.pdf.`

## How to Build the Application

See `<install-folder>\Media_Samples_Guide.pdf.`

## Running the Software

See `<install-folder>\Media_Samples_Guide.pdf.`

The executable file requires the following command-line switches to function properly:

| | |
|---|---|
| h265\|h264\|mpeg2\|vc1\|vp8\|mvc\|jpeg\|capture | Input video type. This is an elementary video stream.<br>The use of option h265 or capture is possible only if corresponding plugins are installed. |
| -i <InputFile> | Input (compressed) video file, name and path. |
| -o <Output> | Specifies output (YUV) video file(s), name and path. With MVC, specify the file name without extension to use as a pattern. The sample creates several output files with names such as “filename_N.yuv” according to the number of views in the MVC stream. |

The following command-line switches are optional:

| | |
|---|---|
| -p <guid> | 32-character hexadecimal guid string which is needed to load plugins. Optional for in-box plugins, required for user-decoder ones (HEVC, f.e.). |
| -path path_to_plugin | Path to decoder plugin (works only in pair with ‘`-p`’ option and requires guid to be specified). |
| -vaapi | Use VAAPI surfaces |
| -r | Enables on-screen rendering support under X Server (requires X Server running) |
| -rdrm | Enables on-screen rendering support thru the framebuffer from the Linux Console mode (requires any graphical server, like X server, to be switched off) |
| -rdrm-<connector> | Same as -rdrm, but enables on-screen rendering support to the monitor connected thru the specified <conector>. Some possible connectors are: DisplayPort, HDMIA, HDMIB, VGA, DVII, DVID, DVIA, eDP and others |
| -perf | Turn on asynchronous flipping for Wayland rendering. |
| -window <x> <y> <w> <h> | Set render window position and size. |
| -hw | Use platform-specific implementation of **SDK** (default) |
| -sw | Use software implementation of **SDK** (platform-specific implementation is used by default) |
| -di <bob or adi> | Enable deinterlacing:BOB or ADI |
| -w | Output width |
| -h | Output height |
| -jpeg_rgb | Set JPEG Color format to RGB4 |
| -jpeg_rotate | Rotate jpeg frame n degrees (90,180,270) |
| -nv12, -i420, -rgb4, -p010, -a2rgb10 | Output color format (native decoder's output format by default ).<br>Note that in case of -i420 option, pipeline uses nv12 color format for output (surfaces format), but during file writing data is converted into i420. |
| -rgb4_fcr | Set pipeline output format and output file format to RGB4 in full color range |
| -low_latency | Configures **Decoding Sample** for low latency mode |
| -calc_latency | Calculate per frame decoding latency |

| | |
|---|---|
| -w | Output width |
| -h | Output height |
| -threads_num | Number of mediasdk task threads |
| -threads_schedtype | Scheduling type of mediasdk task threads |
| -threads_priority | Priority of mediasdk task threads |
| -gpucopy::<on,off> | Enable/disable GPU Copy functionality |
| -timeout | Decode in a loop not less than specific time in seconds. Performs complete input stream decoding on every iteration. Output file frames amount can be bigger than in input due to buffered frames in decoder. Output file is rewrote every iteration. |
| -async | Depth of asynchronous pipeline. Default value is 4. Must be between 1 and 20 |
| -? | Print help |

**Note 1:** You need to have **HEVC** installed to run with h265 codec. In case of HW library (-sw key is not specified) it will firstly try to load HW plugin, in case of failure - it will try SW one if available.

Below are examples of a command-line to execute **Decoding Sample**:

```
$ sample_decode h264 -i input.h264 -o output.yuv -hw
```

Please, also pay attention on "Running the Software" section of `<install-folder>/Media_Samples_Guide.pdf` document where you will find important notes on backend specific usage (drm and x11).

## HEVC Decode Plugins

HEVC codec is implemented as a plugin unlike codecs such as MPEG2 and AVC. We provide 2 implementations of the HEVC decoders: Software (SW) and Hardware (HW).

**Note 1:** The HEVC SW plugin is available in the HEVC package which is part of the Intel® Media Server Studio Professional Edition. You can find the available plugins and their IDs from `$MFX_ROOT/include/mfxplugin.h` file.

**Note 2:** HW plugin for HEVC Decode is supported starting from 6th Generation of Intel CoreTM Processors, Intel® Xeon® E3-1200 and E3-1500 v5 Family with Intel® Processor Graphics 500 Series (codename Skylake).

**Note 3:** Decoding sample loads the HW HEVC decode plugin with HW library and SW decode with SW library by default. You can enforce a plugin to be loaded by specifying the plugin ID using "-p" parameter and hexadecimal GUID.

Examples of using HW HEVC decoder and HW library:

```
$ sample_decode h265 -i input.265 -o output.yuv
$ sample_decode h265 -i input.265 -o output.yuv -hw
```

Examples of SW HEVC decoder usage (the first instance loads SW library, the second one HW library):

```
$ sample_decode h265 -i input.265 -o output.yuv -sw
$ sample_decode h265 -i input.265 -o output.yuv –p
 33a61cb4c27454ca8d85dde757c6f8e
```

## Known Limitations

- **Decoding Sample** does not fully decode some video streams from a networked folder. Instead, copy the input file to local storage prior to decoding.
- **Decoding Sample** renders output in the simplest way. The rendering window does not support time stamps, aspect ratio.
- Decoding Sample cannot render P010 streams in case of SW library usage.
- `-low_latency` and `-calc_latency` options should be used with H.264 streams having exactly 1 slice per frame. Preferable streams for an adequate latency estimate are generated by **Conferencing Sample**. The options are also effective for JPEG* input streams. For all other input formats application would return an error.
- If overlay is not supported by your hardware or software you won't be able to render the decoded MVC stream.
- Application may return error for some MJPEG streams decoded with option –low_latency.
- In case of using HEVC plugin (h265 video type), only software implementation of that plugin is used by default (even if you provide -hw option). To force usage of HEVC HW plugin implementation, please use -p option with proper plugin GUID.
- VP8 HW decoding is not working if system memory is used.
- SW HEVC plugin in 10bit mode cannot be used together with HW library VPP. Although library allows that, this is bad practice because additional per-pixel data shift is required. Please use HW HEVC + HW library or SW HEVC + SW library instead.
- Low latency mode (-low_latency) is not compatible with HEVC decoder.
- Sample may exit with wrong status code (MFX_MORE_SURFACE) and messages

  [ERROR], sts=MFX_ERR_MORE_SURFACE(-11), RunDecoding, Unexpected error!!

  [ERROR], sts=MFX_ERR_MORE_SURFACE(-11), main, Pipeline.RunDecoding failed

  This may occur when number of outputted frames becomes equal to number of frames specified by "-n <frames>" command line parameter while current status is MFX_MORE_SURFACE. No negative side effects present, requested number of frames are properly decoded.
- Sample may not function properly on systems that have a non-Intel VGA controller as the first (primary) because Intel device is not first in the list.

  To workaround this issue, swap names of DRI device files:

  ```
  $ cd /dev && mv card0 tmp && mv card1 card0 && mv tmp card1
  ```

  and do the same for the files `control64/65` and `renderD128/129`

## Legal Information