ISO/IEC JTC 1/SC 29/WG 03 N301

**ISO/IEC JTC 1/SC 29/WG 03  
MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Technologies under Consideration for VDI

**Status:** Approved

**Date of document:** 2021-07-16

**Source:** ISO/IEC JTC 1/SC 29/WG 03

**No. of pages:** 12 (with cover page)

**Email of Convenor:** young.L @ samsung . com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N301**

**July 2021, Virtual**

|  |  |
| --- | --- |
| **Title** | **Technologies under Consideration for VDI** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **20555** |

# Introduction

This document collects technologies under consideration for the development of ISO/IEC 23090-13, a.k.a. MPEG-I part 13: Video Decoding Interfaces for Immersive Media.

# Contents

[Introduction 1](#_Toc77350290)

[Contents 1](#_Toc77350291)

[1 Relevant W3C Recommendation and their relationship with VDI (from m56783) 2](#_Toc77350292)

[1.1 Media Source Extensions [1] 2](#_Toc77350293)

[1.2 Metadata API for Media Resources [2] 3](#_Toc77350294)

[1.3 Media Capabilities [3] 3](#_Toc77350295)

[1.4 HTMLVideoElement.requestVideoFrameCallback() [4] 4](#_Toc77350298)

[1.5 MediaStreamTrack Content Hits [5] 5](#_Toc77350300)

[1.6 WebCodecs [6] 5](#_Toc77350302)

[2 Khronos Vulkan Video extension (from m57544) 6](#_Toc77350304)

[2.1 Proposed design 6](#_Toc77350305)

[2.2 Proposed Text 7](#_Toc77350306)

[2.3 Mapping on Vulkan® 7](#_Toc77350314)

[2.3.1 Overview 7](#_Toc77350315)

[2.4 Mapping of VDI functions 7](#_Toc77350316)

[2.4.1 Overview 7](#_Toc77350317)

[2.4.2 queryCurrentAggregateCapabilities 7](#_Toc77350318)

[2.4.3 getInstance 8](#_Toc77350319)

[2.4.4 setConfig 8](#_Toc77350320)

[2.4.5 getParameter and setParameter 9](#_Toc77350321)

[3 References 9](#_Toc77350322)

# Relevant W3C Recommendation and their relationship with VDI (from m56783)

# Media Source Extensions [1]

This specification extends [HTMLMediaElement](https://www.w3.org/TR/html51/semantics-embedded-content.html#htmlmediaelement-htmlmediaelement) [[*HTML51*](https://www.w3.org/TR/media-source/#bib-HTML51)] to allow JavaScript to generate media streams for playback. Allowing JavaScript to generate streams facilitates a variety of use cases like adaptive streaming and time shifting live streams.

### Relevance to VDI

MSE is relevant because it is meant to facilitate the manipulating the HTML media element in a buffer level - to facilitate advanced playback and buffering functionalities.

Most functionalities of MSE are around the **MediaSource** object that is acting as a souce for the HTMLMediaElement. By itself it does not have any way to interpret the content, as such it requires a SourceBuffer object to be attached to it. Therefore, most of the methods and events of the MediaSource are designed with the purpose of “interfacing” the attached SourceBuffer objects with the HTML media element.

For the sake of completeness, the events defined for MediaSource are attached bellow:

| **Event name** | **Interface** | **Dispatched when...** |
| --- | --- | --- |
| *sourceopen* | Event | [readyState](https://www.w3.org/TR/media-source/#dom-readystate) transitions from ["closed"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.closed) to ["open"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.open) or from ["ended"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.ended) to ["open"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.open). |
| *sourceended* | Event | [readyState](https://www.w3.org/TR/media-source/#dom-readystate) transitions from ["open"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.open) to ["ended"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.ended). |
| *sourceclose* | Event | [readyState](https://www.w3.org/TR/media-source/#dom-readystate) transitions from ["open"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.open) to ["closed"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.closed) or ["ended"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.ended) to ["closed"](https://www.w3.org/TR/media-source/#idl-def-ReadyState.closed). |

Since the **SourceBuffer** object is the one handling the streams prior going to the MediaSource, it is more relevant to the scope of VDI. SourceBuffer takes as inputs **media segments** – which in the scope of this specification consist of a series of coded frames with a duration assigned. Most functionality of a Source Buffer is achieved by updating the objects with new media segments, and changing the way they are updated (if needed) to allow non-timestamp-based representations of frames. To provide a better overview of the SourceBuffer, following is the interface as defined in the specification:

interface SourceBuffer : EventTarget {

attribute [AppendMode](https://www.w3.org/TR/media-source/#idl-def-appendmode) [mode](https://www.w3.org/TR/media-source/#dom-sourcebuffer-mode);

readonly attribute [boolean](https://www.w3.org/TR/WebIDL-1/#idl-boolean) [updating](https://www.w3.org/TR/media-source/#dom-sourcebuffer-updating);

readonly attribute [TimeRanges](https://www.w3.org/TR/html51/semantics-embedded-content.html#timeranges-timeranges) [buffered](https://www.w3.org/TR/media-source/#dom-sourcebuffer-buffered);

attribute [double](https://www.w3.org/TR/WebIDL-1/#idl-double) [timestampOffset](https://www.w3.org/TR/media-source/#dom-sourcebuffer-timestampoffset);

readonly attribute [AudioTrackList](https://www.w3.org/TR/html51/semantics-embedded-content.html#audiotracklist-audiotracklist) [audioTracks](https://www.w3.org/TR/media-source/#dom-sourcebuffer-audiotracks);

readonly attribute [VideoTrackList](https://www.w3.org/TR/html51/semantics-embedded-content.html#videotracklist-videotracklist) [videoTracks](https://www.w3.org/TR/media-source/#dom-sourcebuffer-videotracks);

readonly attribute [TextTrackList](https://www.w3.org/TR/html51/semantics-embedded-content.html#texttracklist-texttracklist) [textTracks](https://www.w3.org/TR/media-source/#dom-sourcebuffer-texttracks);

attribute [double](https://www.w3.org/TR/WebIDL-1/#idl-double) [appendWindowStart](https://www.w3.org/TR/media-source/#dom-sourcebuffer-appendwindowstart);

attribute [unrestricted double](https://www.w3.org/TR/WebIDL-1/#idl-unrestricted-double) [appendWindowEnd](https://www.w3.org/TR/media-source/#dom-sourcebuffer-appendwindowend);

attribute [EventHandler](https://www.w3.org/TR/html51/webappapis.html#events-event-handlers) [onupdatestart](https://www.w3.org/TR/media-source/#dom-sourcebuffer-onupdatestart);

attribute [EventHandler](https://www.w3.org/TR/html51/webappapis.html#events-event-handlers) [onupdate](https://www.w3.org/TR/media-source/#dom-sourcebuffer-onupdate);

attribute [EventHandler](https://www.w3.org/TR/html51/webappapis.html#events-event-handlers) [onupdateend](https://www.w3.org/TR/media-source/#dom-sourcebuffer-onupdateend);

attribute [EventHandler](https://www.w3.org/TR/html51/webappapis.html#events-event-handlers) [onerror](https://www.w3.org/TR/media-source/#dom-sourcebuffer-onerror);

attribute [EventHandler](https://www.w3.org/TR/html51/webappapis.html#events-event-handlers) [onabort](https://www.w3.org/TR/media-source/#dom-sourcebuffer-onabort);

[void](https://www.w3.org/TR/WebIDL-1/#idl-void) [appendBuffer](https://www.w3.org/TR/media-source/#dom-sourcebuffer-appendbuffer)([BufferSource](https://www.w3.org/TR/WebIDL-1/" \l "common-BufferSource) data);

[void](https://www.w3.org/TR/WebIDL-1/#idl-void) [abort](https://www.w3.org/TR/media-source/#dom-sourcebuffer-abort)();

[void](https://www.w3.org/TR/WebIDL-1/#idl-void) [remove](https://www.w3.org/TR/media-source/#dom-sourcebuffer-remove)([double](https://www.w3.org/TR/WebIDL-1/#idl-double) start, [unrestricted double](https://www.w3.org/TR/WebIDL-1/#idl-unrestricted-double) end);

};

### Conclusion

It is relevant keep monitoring the MSE for future developments and when VDI matures to contact W3C proponents for possible technical alignment of features.

# Metadata API for Media Resources [2]

**Abstract**

This specification defines an API to access metadata information related to media resources on the Web. The overall purpose is to provide developers with a convenient access to metadata information stored in different metadata formats. The API provides means to access the set of metadata properties defined in the [Ontology for Media Resources 1.0](http://www.w3.org/TR/mediaont-10/) specification. These [properties](http://www.w3.org/TR/mediaont-10/#core-property-definitions) are used as a pivot vocabulary in this API. The core of this specification is the definition of API interfaces for retrieving metadata information in synchronous and asynchronous modes. It also defines interfaces for structured return types along with the specification of the behavior of an API implementation.

### Relevance to VDI

Even though this specification precedes MSE in conception it was recently revived with promises for implementations and testing by browser vendors. The main goal of this API is to interface media metadata (mostly with semantical properties), however, there is consideration for technical properties, with current examples including values for FrameRate, AverageBitRate, NumTracks etc. As such it might be useful in the future to accommodate technical properties that would enable/optimize adoption of VDI (e.g. recommended number of decoders).

### Conclusion

For now, Metadata API is not interesting for VDI, but in the (unlikely?) case that it gets traction in the future we might want to use it for VDI signaling.

# Media Capabilities [3]

This specification intends to provide APIs to allow websites to make an optimal decision when picking media content for the user. The APIs will expose information about the decoding and encoding capabilities for a given format but also output capabilities to find the best match based on the device’s display.



### Relevance to VDI

This is an active (and new) specification that is probably the most relevant to VDI – alongside MSE. It seems to be overlapping with the Metadata API in that it can signal supported/available bitrates, codecs etc. with a focus on decoder and encoder capabilities. The high level of the interface is defined as following:

### Media Capabilities Interface

[[Exposed](https://heycam.github.io/webidl/#Exposed)=(Window, Worker)]

interface *MediaCapabilities* {

[[NewObject](https://heycam.github.io/webidl/" \l "NewObject)] [Promise](https://heycam.github.io/webidl/#idl-promise)<[MediaCapabilitiesDecodingInfo](https://www.w3.org/TR/media-capabilities/" \l "dictdef-mediacapabilitiesdecodinginfo)> *decodingInfo*([MediaDecodingConfiguration](https://www.w3.org/TR/media-capabilities/" \l "dictdef-mediadecodingconfiguration) *configuration*);

[[NewObject](https://heycam.github.io/webidl/" \l "NewObject)] [Promise](https://heycam.github.io/webidl/#idl-promise)<[MediaCapabilitiesEncodingInfo](https://www.w3.org/TR/media-capabilities/" \l "dictdef-mediacapabilitiesencodinginfo)> *encodingInfo*([MediaEncodingConfiguration](https://www.w3.org/TR/media-capabilities/" \l "dictdef-mediaencodingconfiguration) *configuration*);

};

With the example MediaDecodingType having the following types:

### MediaDecodingType

enum *MediaDecodingType* {

["file"](https://www.w3.org/TR/media-capabilities/#dom-mediadecodingtype-file),

["media-source"](https://www.w3.org/TR/media-capabilities/#dom-mediadecodingtype-media-source),

["webrtc"](https://www.w3.org/TR/media-capabilities/#dom-mediadecodingtype-webrtc)

};

And the video configuration currently holding the following properties:

dictionary *VideoConfiguration* {

required [DOMString](https://heycam.github.io/webidl/#idl-DOMString) [contentType](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-contenttype);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [width](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-width);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [height](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-height);

required [unsigned long long](https://heycam.github.io/webidl/#idl-unsigned-long-long) [bitrate](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-bitrate);

required [double](https://heycam.github.io/webidl/#idl-double) [framerate](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-framerate);

[boolean](https://heycam.github.io/webidl/#idl-boolean) [hasAlphaChannel](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-hasalphachannel);

[HdrMetadataType](https://www.w3.org/TR/media-capabilities/#enumdef-hdrmetadatatype) [hdrMetadataType](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-hdrmetadatatype);

[ColorGamut](https://www.w3.org/TR/media-capabilities/#enumdef-colorgamut) [colorGamut](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-colorgamut);

[TransferFunction](https://www.w3.org/TR/media-capabilities/#enumdef-transferfunction) [transferFunction](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-transferfunction);

[DOMString](https://heycam.github.io/webidl/#idl-DOMString) [scalabilityMode](https://www.w3.org/TR/media-capabilities/#dom-videoconfiguration-scalabilitymode);

};

### Conclusion

Media Capabilities in its present form is not useful for VDI, but since this is a new effort, driven by browser vendors (Google, Microsoft), it makes sense to monitor the developments.

# HTMLVideoElement.requestVideoFrameCallback() [4]

<video>.requestVideoFrameCallback() allows web authors to be notified when a frame has been presented for composition.



### Relevance to VDI

This is a new feature which consists of a callback for the video element, that has been decided to be actually implemented (at least for testing) from at least two major browsers (Chrome and Firefox). It fires a callback as soon as a video frame has been decoded and it can be set for single or multiple frames. *It is fired before the requestAnimationFrame callback, which is a window-level callback responsible for refresh the current browser display, as such it makes the decoded frame available prior to being render thus giving time and space for analysis/processing.* Even though in the current version of the spec this is a best-effort feature, thus it might not always be the case. Alongside with the decoded frame (mentioned as “media pixels” in the spec) some metadata are provided as following:

### VideoFrameMetadata

dictionary *VideoFrameMetadata* {

required [DOMHighResTimeStamp](https://www.w3.org/TR/hr-time-2/#dom-domhighrestimestamp) [presentationTime](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-presentationtime);

required [DOMHighResTimeStamp](https://www.w3.org/TR/hr-time-2/#dom-domhighrestimestamp) [expectedDisplayTime](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-expecteddisplaytime);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [width](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-width);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [height](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-height);

required [double](https://heycam.github.io/webidl/#idl-double) [mediaTime](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-mediatime);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [presentedFrames](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-presentedframes);

[double](https://heycam.github.io/webidl/#idl-double) [processingDuration](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-processingduration);

[DOMHighResTimeStamp](https://www.w3.org/TR/hr-time-2/#dom-domhighrestimestamp) [captureTime](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-capturetime);

[DOMHighResTimeStamp](https://www.w3.org/TR/hr-time-2/#dom-domhighrestimestamp) [receiveTime](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-receivetime);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [rtpTimestamp](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-rtptimestamp);

};

From the aforementioned metadata it is worth noticing the processingDuration which is defined as following:

*processingDuration*, of type [double](https://heycam.github.io/webidl/#idl-double)

The elapsed duration in seconds from submission of the encoded packet with the same presentation timestamp (PTS) as this frame (e.g. same as the [mediaTime](https://wicg.github.io/video-rvfc/#dom-videoframemetadata-mediatime)) to the decoder until the decoded frame was ready for presentation.

In addition to decoding time, may include processing time. E.g., YUV conversion and/or staging into GPU backed memory.

SHOULD be present. In some cases, user-agents might not be able to surface this information since portions of the media pipeline might be owned by the OS.

### Conclusion

Even if it is just for the processingDuration, this feature can be very useful for adopting VDI in a browser environment since it can give input for reconfiguring the decoders and fine-tuning the decoding process.

# MediaStreamTrack Content Hits [5]

This specification extends [MediaStreamTrack](https://www.w3.org/TR/mediacapture-streams/#dom-mediastreamtrack) to provide a media-content hint attribute. This optional hint permits [MediaStreamTrack](https://www.w3.org/TR/mediacapture-streams/#dom-mediastreamtrack) consumers such as RTCPeerConnection (defined in [*[webrtc](https://w3c.github.io/mst-content-hint/" \l "bib-webrtc" \o "WebRTC 1.0: Real-Time Communication Between Browsers)*]) or MediaRecorder (defined in [*[mediastream-recording](https://w3c.github.io/mst-content-hint/" \l "bib-mediastream-recording" \o "MediaStream Recording)*]) to encode or process track media with methods more appropriate to the type of content that is being consumed.

Adding a media-content hint provides a way for a web application to help track consumers make more informed decision of what encoder parameters and processing algorithms to use on the consumed content.



### Relevance to VDI and Conclusion

At its current form this feature is not essential for VDI since it is focusing on the actual content type of the media stream (e.g. for video it can signal “motion”, “detail”, “text”…). We mention it here purely for the sake of completeness because it will probably be adopted at least by Chrome and Firefox.

# WebCodecs [6]

This specification defines interfaces to codecs for encoding and decoding of audio and video.

This specification does not specify or require any particular codec or method of encoding or decoding. The purpose of this specification is to provide JavaScript interfaces to implementations of existing codec technology developed elsewhere. Implementers may support any combination of codecs or none at all.



### Relevance to VDI

This is actually about control of the decoding process (note: in the specification **codec** is defined as follows: Refers generically to an instance of AudioDecoder, AudioEncoder, VideoDecoder, or VideoEncoder.). As such it mentions that:

### Codec Processing Model

### Background

This section is non-normative.

The codec interfaces defined by the specification are designed such that new codec tasks may be scheduled while previous tasks are still pending. For example, web authors may call decode() without waiting for a previous decode() to complete. This is achieved by offloading underlying codec tasks to a separate thread for parallel execution.

This section describes threading behaviors as they are visible from the perspective of web authors. Implementers may choose to use more or less threads as long the externally visible behaviors of blocking and sequencing are maintained as follows.

Regarding the processing model the following are mentioned:

### Control Thread and Codec Thread

All steps in this specificaiton will run on either a [control thread](https://www.w3.org/TR/webcodecs/#control-thread) or a [codec thread](https://www.w3.org/TR/webcodecs/#codec-thread).

The *control thread* is the thread from which authors will construct a [codec](https://www.w3.org/TR/webcodecs/#codec) and invoke its methods. Invoking a codec’s methods will typically result in the creation of [control messages](https://www.w3.org/TR/webcodecs/#control-messages) which are later executed on the [codec thread](https://www.w3.org/TR/webcodecs/#codec-thread). Each [global object](https://html.spec.whatwg.org/multipage/webappapis.html#global-object) has a separate control thread.

The *codec thread* is the thread from which a [codec](https://www.w3.org/TR/webcodecs/#codec) will [dequeue](https://infra.spec.whatwg.org/#queue-dequeue) [control messages](https://www.w3.org/TR/webcodecs/#control-messages) and execute their steps. Each [codec](https://www.w3.org/TR/webcodecs/#codec) instance has a separate codec thread. The lifetime of a codec thread matches that of its associated [codec](https://www.w3.org/TR/webcodecs/#codec) instance.

This will work with interfaces separately for audio and video, with the VideoDecore Interface (https://www.w3.org/TR/webcodecs/#videodecoder-interface) being obviously the most relevant to this group. The VideoDecoder Interface can be set using VideoDecoderInit (setting output and error handling), parameterized with VideoDecoderConfig and having as input EncodedVideoChuck and as output a VideoFrameOutputCallback. The current configuration parameters are the following:

### VideoDecoderConfig

dictionary *VideoDecoderConfig* {

required [DOMString](https://heycam.github.io/webidl/#idl-DOMString) [codec](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-codec);

[BufferSource](https://heycam.github.io/webidl/#BufferSource) [description](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-description);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [codedWidth](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-codedwidth);

required [unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [codedHeight](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-codedheight);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [cropLeft](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-cropleft);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [cropTop](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-croptop);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [cropWidth](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-cropwidth);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [cropHeight](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-cropheight);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [displayWidth](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-displaywidth);

[unsigned long](https://heycam.github.io/webidl/#idl-unsigned-long) [displayHeight](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-displayheight);

[HardwareAcceleration](https://www.w3.org/TR/webcodecs/#enumdef-hardwareacceleration) [hardwareAcceleration](https://www.w3.org/TR/webcodecs/#dom-videodecoderconfig-hardwareacceleration) = "allow";

};

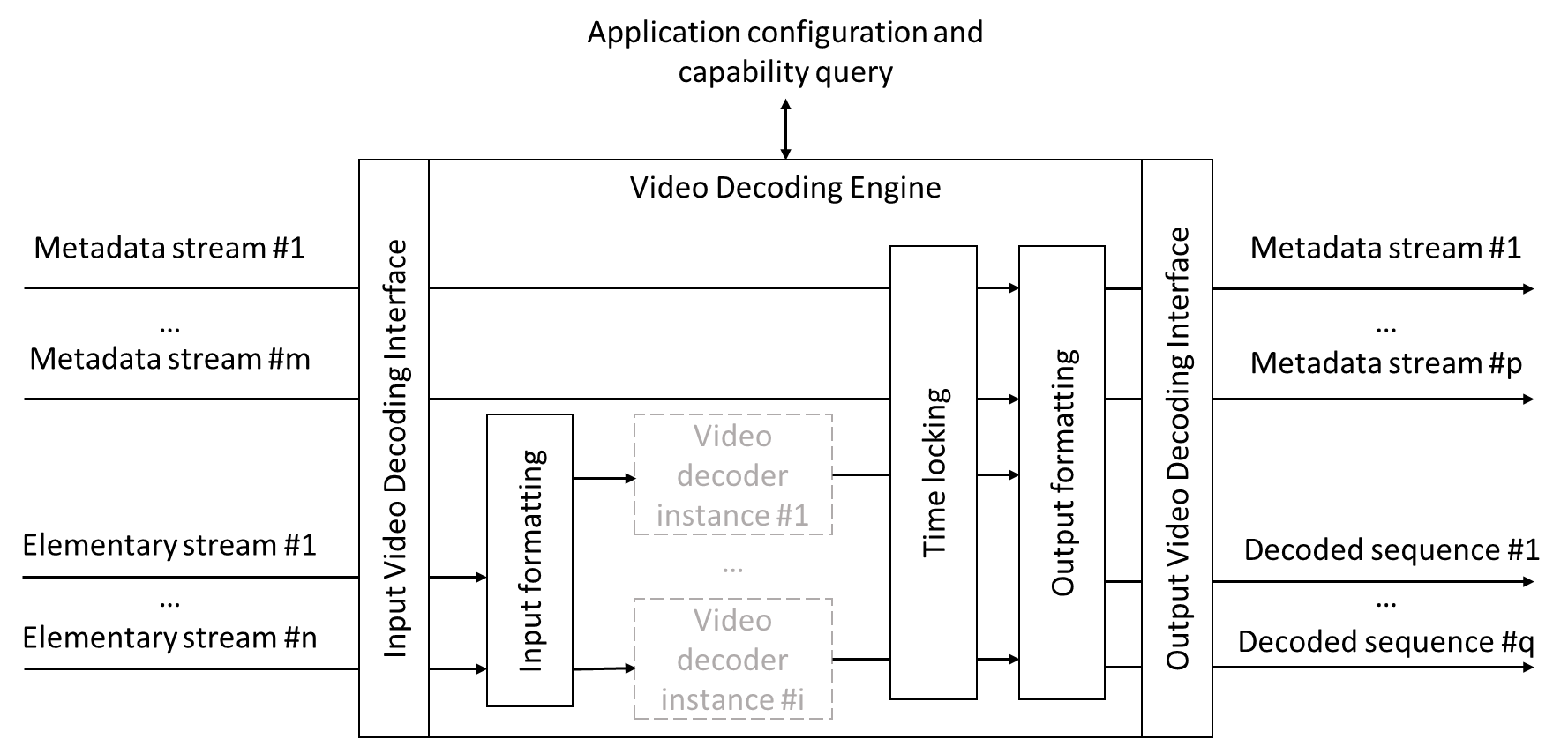
# Khronos Vulkan Video extension (from m57544)

# Proposed design

As seen above, a VK Video Session operates a single decoding session on a single layer. As a result, one can envision two ways of mapping VDI onto Vulkan:

1. Extending a VK Video Session to operate on multiple video decoder instances.
2. Creating a wrapper around multiple VK Video Sessions.

In order to avoid redefining core definitions of the VK Video API, it seems more desirable to opt for option 2 which means mapping a VK Video Session onto a video decoder instance as follows:



VK Video Session

VK Video Session

Figure 8 - Proposed mapping of a VK Video Session on VDI

Therefore, the VDI concept of grouping video decoding instances will correspond to grouping multiple VK Video Sessions.

# Proposed Text



# Mapping on Vulkan®

# Overview

Vulkan® Video (VK) is an extension of the Vulkan API which defines standardised functions offered by GPUs. This extension provides interfaces for an application to leverage hardware decoding and encoding capabilities present on GPUs. For more information about the Vulkan API, please refer to [Vulkan-link].

# Mapping of VDI functions

# Overview

The mapping of VDI function on VK is summarised by Table XYZ.

Table XYZ - Summary of VDI function mapping on VK

|  |  |
| --- | --- |
| VDI Functionality | Vulkan mapping |
| **queryCurrent AggregateCapabilities** | New function |
| **getInstance** (grouping) | Extending VkVideoSessionCreateInfoKHR with a group ID attribute. Call of existing vkCreateVideoSessionKHR(). |
| **setConfig (buffer configuration)** | Mapping on existing structures. |
| **getParameter** and **setParameter** | New structure |

# queryCurrentAggregateCapabilities

VkResult vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG(

VkPhysicalDevice physicalDevice,

**const** VkVideoProfileKHR\* pVideoProfile,

VkCurrentVideoCapabilitiesMPEG\* pCapabilities);

**typedef** **struct** VkCurrentVideoCapabilitiesMPEG {

VkStructureType sType;

**void**\* pNext;

uint32\_t capInstances;

uint64\_t capBufferMemory;

uint32\_t capBitrate;

uint32\_t capMaxSamplesSecond;

VkPerformancePointMPEG capMaxPerformancePoint;

} VkCurrentVideoCapabilitiesMPEG;

**typedef** **struct** VkPerformancePointMPEG {

VkStructureType sType;

**void**\* pNext;

uint32\_t pictureRate;

uint32\_t height;

uint32\_t width;

uint32\_t bitdepth;

} VkPerformancePointMPEG;

# getInstance

VkVideoSessionCreateInfoGroupingMPEG extending the existing VkVideoSessionCreateInfoKHR.

**typedef** **struct** VkVideoSessionCreateInfoGroupingMPEG {

VkStructureType sType;

**const** **void**\* pNext;

uint32\_t groupId;

} VkVideoSessionCreateInfoKHR;

# setConfig

The existing VkVideoSessionCreateInfoKHR and VkVideoPictureResourceKHR map onto setConfig.

**typedef** **struct** VkVideoSessionCreateInfoKHR {

VkStructureType sType;

**const** **void**\* pNext;

uint32\_t queueFamilyIndex;

VkVideoSessionCreateFlagsKHR flags;

**const** VkVideoProfileKHR\* pVideoProfile;

VkFormat pictureFormat;

VkExtent2D maxCodedExtent;

VkFormat referencePicturesFormat;

uint32\_t maxReferencePicturesSlotsCount;

uint32\_t maxReferencePicturesActiveCount;

} VkVideoSessionCreateInfoKHR;

**typedef** **struct** VkVideoProfileKHR {

VkStructureType sType;

**void**\* pNext;

VkVideoCodecOperationFlagBitsKHR videoCodecOperation;

VkVideoChromaSubsamplingFlagsKHR chromaSubsampling;

VkVideoComponentBitDepthFlagsKHR lumaBitDepth;

VkVideoComponentBitDepthFlagsKHR chromaBitDepth;

} VkVideoProfileKHR;

**typedef** **struct** VkVideoPictureResourceKHR {

VkStructureType sType;

**const** **void**\* pNext;

VkOffset2D codedOffset;

VkExtent2D codedExtent;

uint32\_t baseArrayLayer;

VkImageView imageViewBinding;

} VkVideoPictureResourceKHR;

|  |  |
| --- | --- |
| VDI | Vulkan Video |
| sample\_format | pictureFormat |
| sample\_type | pictureFormat |
| sample\_stride | codedExtent |
| line\_stride | codedExtent |
| buffer\_offset | codedOffset |

Note: Mapping to be refined

# getParameter and setParameter

**typedef** **struct** VkVideoSessionOutputParameterMPEG {

VkStructureType sType;

**const** **void**\* pNext;

VkFlag subframeOutput;

void\* metadataCallback;

VkFlag outputCrop;

VkFlag subpictureOutput;

uint32\_t maxOfftimeJitter;

} VkVideoSessionOutputParameterMPEG;

# References

1. Media Source Extensions, W3C Draft, <https://www.w3.org/TR/media-source/>
2. Metadata API For Media Resources, W3C Draft, <https://www.w3.org/TR/mediaont-api-1.0>
3. Media Capabilities, W3C Draft, <https://www.w3.org/TR/media-capabilities/>
4. HTMLVideoElement.requestVideoFrameCallback(), W3C Draft, <https://wicg.github.io/video-rvfc/>
5. MediaStreamTrack Content Hits, W3C Draft,<https://w3c.github.io/mst-content-hint/>
6. WebCodecs, W3C Draft, <https://www.w3.org/TR/webcodecs/>