

**ISO/IEC JTC 1/SC 29/WG 11**

**Coding of moving pictures and audio**

**Convenorship: Japan (JISC)**

**ISO/IEC JTC 1/SC 29/WG 11 N 19443**

**Document type:** Approved WG 11 document

**Title:** WD on Video Decoding Interface for Immersive Media

**Status:** Approved

**Date of document:** 2020-07-12

**Source:** Convenor, ISO/IEC JTC 1/SC 29/WG 11

**No. of pages:** 25

**Email of convenor:** ostermann@tnt.uni-hannover.de

**Committee URL:** http://isotc.iso.org/livelink/livelink/open/jtc1sc29

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 11 N** **19443**

**Online – June 2020**

|  |  |
| --- | --- |
| **Source:** | **Systems SubGroup** |
| **Title:** | **WD on Video Decoding Interface for Immersive Media** |

**WD on Video Decoding Interface for Immersive Media**

**ISO 23090-13:202x(E)**

ISO TC 1 SC 19/WG 1

Secretariat: XXXX

**Information technology — Coded representation of immersive media — Part 13: Video Decoding Interface for Immersive Media**

WD stage

**Warning for WDs and CDs**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

© ISO 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO’s member body in the country of the requester.

ISO copyright office

CP 401 • Ch. de Blandonnet 8

CH-1214 Vernier, Geneva

Phone: +41 22 749 01 11

Fax: +41 22 749 09 47

Email: copyright@iso.org

Website: www.iso.org

Published in Switzerland

Contents

[Foreword vi](#_Toc45461607)

[Introduction vii](#_Toc45461608)

[1 Scope 1](#_Toc45461609)

[2 Normative references 1](#_Toc45461610)

[3 Terms and definitions 1](#_Toc45461611)

[4 Timeline 2](#_Toc45461612)

[5 Motivations 2](#_Toc45461613)

[6 Video Decoding Engine 2](#_Toc45461614)

[6.1 General 2](#_Toc45461615)

[6.2 Input Video Decoding Interface 4](#_Toc45461616)

[6.3 Output Video Decoding Interface 4](#_Toc45461617)

[6.4 Control interface to the Video Decoding Interface 4](#_Toc45461618)

[6.4.1 Functions 4](#_Toc45461619)

[6.5 Examples of Video Decoding Engine instantiations 7](#_Toc45461620)

[6.5.1 OpenMAX™ 7](#_Toc45461621)

[6.5.2 MSE 10](#_Toc45461622)

[7 Video decoder interface 10](#_Toc45461623)

[7.1 Operations on input elementary streams 11](#_Toc45461624)

[7.1.1 Conventions 11](#_Toc45461625)

[7.1.2 Concepts 11](#_Toc45461626)

[7.1.3 General 11](#_Toc45461627)

[7.1.4 Inserting video objects 12](#_Toc45461628)

[7.2 Instantiation for ISO/IEC 23090-2 Versatile Video Codec 14](#_Toc45461629)

[7.2.1 General 14](#_Toc45461630)

[7.3 Instantiation for HEVC 14](#_Toc45461631)

[7.3.1 General 14](#_Toc45461632)

[Annex A (normative) Control Interface IDL Syntax 16](#_Toc45461633)

[A.1 General 16](#_Toc45461634)

[A.2 Interface definition 16](#_Toc45461635)

[Bibliography 18](#_Toc45461636)

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see [www.iso.org/directives](https://www.iso.org/directives-and-policies.html)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](https://www.iso.org/iso-standards-and-patents.html)).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see [www.iso.org/iso/foreword.html](https://www.iso.org/foreword-supplementary-information.html).

This document was prepared by Technical Committee ISO/IEC/JTC 1 *Information technology*, Subcommittee SC 29, Coding of audio, picture, multimedia and hypermedia information.

This is the first edition published under ISO 23090-13:202x.

A list of all parts in the ISO 23090 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user’s national standards body. A complete listing of these bodies can be found at [www.iso.org/members.html](https://www.iso.org/members.html).

Introduction

Type text.

*Identification of patent holders, if any.*

Information technology — Coded representation of immersive media — Part 13: Multi-Decoder Video Interface for Immersive Media

# Scope

This document specifies the control, input, and output interfaces of a video decoding engine as well as the operations that can be performed by this video decoding engine: input formatting on elementary streams, time locking of decoded sequences and metadata streams, output formatting of decoded sequences and metadata streams, and the API for the application to control these operations.

# Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23008-2 *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding*

ISO/IEC 23090-3 *Information technology — Coded representation of immersive media — Part 3: Versatile video coding*

ISO/IEC 19516 *Information technology — Object management group — Interface definition language (IDL) 4.2*

# Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1

video object

independently decodable substream of a video elementary stream

Note 1 to entry: Examples of data type are visual textures, depth maps, occupancy maps, etc…

3.2

video object identifier

an integer identifying a video object

# Timeline

The expected timeline for this specification is:

* July 2021 (FDIS) as part of MPEG-I Phase 2a
* CD in October 2020

# Motivations

The goal of this specification is to provide enablers to integrate efficiently MPEG-I technologies into existing media ecosystems such as hardware video decoding platform, GPU rendering pipeline, etc.

To this end, the following structure is suggested:

1. Define a generic part of the video decoding interfaces which is codec-agnostic:
   1. Define bindings of this video decoding interface with example technologies
   2. Use the Khronos OpenMAX™ as a first binding to progress the work
2. Define the video decoder interface in a codec-agnostic way (as defined in the "Requirements on Immersive Media Access and Delivery" N18654) and the associated operations on video objects (from the "Thoughts on Immersive Media Decoding Interface for VVC" N18615):
   1. Define a binding for VVC using the layer feature of VVC to instantiate the concept of video objects
   2. Define a binding for HEVC using a tile/slice to instantiate the concept of video objects
3. Develop software to illustrate and verify the successful integration with external ecosystems:
   1. A sample software leveraging the video decoding interface based on Khronos OpenMAX™
   2. A Test Model for manipulating video elementary bitstreams following the N18615

# Video Decoding Engine

## General

The Video Decoding Engine (VDE) enables the decoding, the synchronisation and the formatting of one or more elementary streams fed through the Input Video Decoding Interface (IVDI) of the VDE and provided to the subsequent elements of the rendering pipeline via the Output Video Decoding Interface (OVDI). The VDE extracts and merges independently coded regions of a set of input elementary streams and generates new set of elementary streams fed to the video decoder instances running inside the engine through the input formatting function. The VDE would execute a merging operation on the elementary streams when the number of video decoder instances is lower than the number of input elementary streams. Conversely, the VDE would execute an extraction operation when the number of video decoder instances is greater than the number of input elementary streams.

Figure 1 presents a possible architecture for the VDE and the associated IVDI and OVDI interfaces.

Figure 2 depicts an architecture for handling multiple video decoder instances on a single hardware platform. In this scenario, one or more video decoder instances running on the same video decoder hardware engine are exposed to the application layer in form of several decoder instances each with their own interfaces.

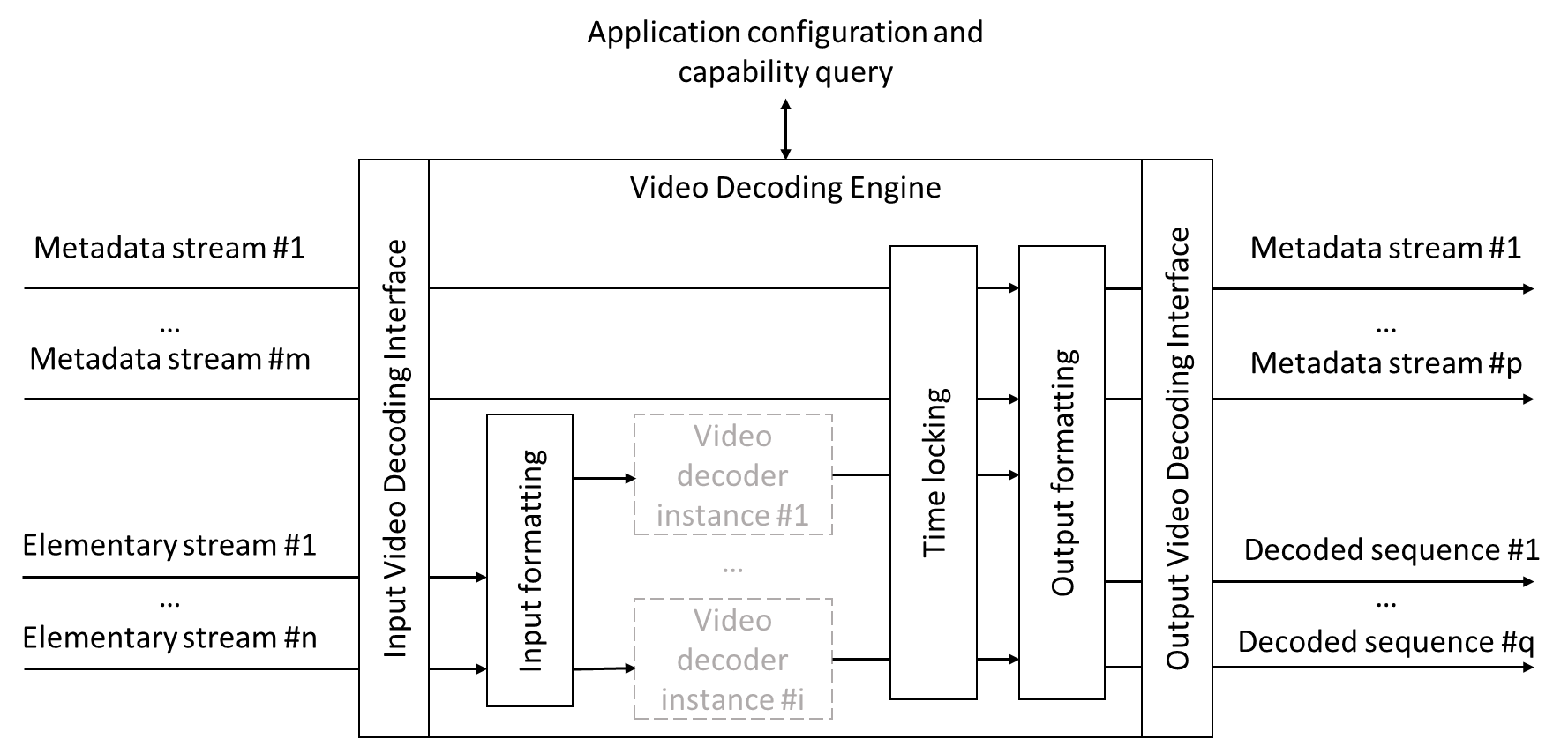


Figure 1 - Video Decoding Engine and interfaces

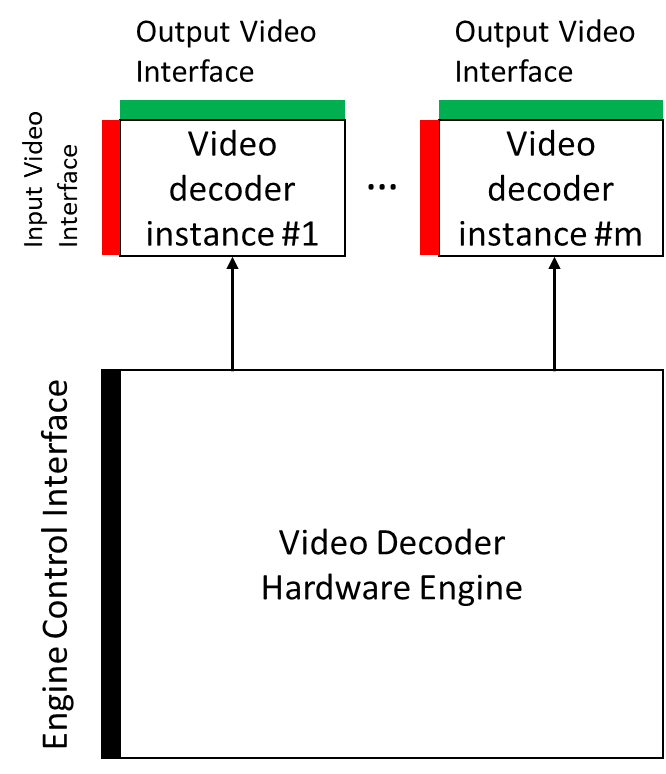


Figure 2 - Relation between video decoder instances and video decoder hardware engine

## Input Video Decoding Interface

The input of the VDE comprises the following comprises:

* n elementary streams
* m metadata streams

## Output Video Decoding Interface

The output of the VDE comprises the following comprises:

* p decoded sequences
* q metadata streams

These two output stream types may be provided in form of multiplexed output buffers, including both decoded media data and its associated metadata.

## Control interface to the Video Decoding Interface

### Functions

The following functions are defined using the IDL syntax specified in ISO/IEC 19516 Information technology — Object management group — Interface definition language (IDL) 4.2.

#### queryCurrentAggregateCapabilities

##### Declaration

AggregateCapabilities queryCurrentAggregateCapabilities(  
 string component\_name,  
 int flags)

##### Definition

The following function can be used by the application to query the instantaneous aggregate capabilities of a decoder platform for a specific codec component.

The component\_name provides the name of the media component of the decoding platform for which the query applies. The name "All" may be used to indicate that the query is not for a particular component but rather is for all components of the decoding platform.

The following capability flags can be queried separately or in a single function call:

* CAP\_INSTANCES: this flag indicates the maximum number of decoder instances that can be instantiated at this instant for the provided decoder component.
* CAP\_BUFFER\_MEMORY: this flag indicates the maximum available buffer size that can be allocated at this instant on the decoder platform for buffer exchange with components of the media decoding platform. Note that this value is independent from any media components and applies globally to the decoding platform. The value shall be expressed in bytes.
* CAP\_BITRATE: This flag queries the maximum coded bitrate in bits per second that the queried component is able to process at this instant.
* CAP\_MAX\_SAMPLES\_SECOND: This flag queries the maximum number of samples per second that the queried component is able to process at this instant.
* CAP\_MAX\_PERFORMANCE\_POINT: This flag is used to query the maximum performance point of a bitstream that can be decoded by the indicated component in a new instance of that decoder component. A performance point has the following parameters:
  + PICTURE\_RATE: the picture rate of the maximum performance point in pictures per second.
  + HEIGHT: the height in luma samples of the maximum performance point
  + WIDTH: the width in luma samples of the maximum performance point
  + BITDEPTH: the bit depth of the luma samples of the maximum performance point

NOTE Each parameter of the max performance point does not necessarily represent the maximum in that dimension. It is the combination of all dimensions that constitutes the maximum performance point.

#### getInstance

##### Declaration

int getInstance(string component\_name, int group\_id=-1)

##### Definition

The result of a successful call to the getInstance function call shall contain the identifier of the instance and the group\_id that is assigned or created for this instance, if one was requested. The default is that the decoder instance does not belong to any already established group but creates a new group.

#### setConfig

##### Declaration

boolean setConfig(  
 int instance\_id,  
 ConfigParameters config\_parameters,  
 ConfigDataParameters config\_data\_parameters  
)

##### Definition

The setConfig function may be called with a new parameter "CONFIG\_OUTPUT\_BUFFER", in which case it provides the format of the output buffer.

The format of the buffer shall contain the following parameters:

* sample\_format: indicates the format of each sample, which can be a scalar, a 2D vector, a 3D vector, or a 4D vector.
* sample\_type: indicates the type of each component of the sample.
* sample\_stride: indicates the number of bytes between 2 consecutive samples of this output.
* line\_stride: indicates the number of bytes between the first byte of one line and the first byte of the following line of this output.
* buffer\_offset: indicates the offset into the output buffer, starting from which the output frame should be written

#### getParemeter and setParameter

##### Declaration

any getParameter(  
 int instance\_id,  
 ExtParameters ext\_parameters,  
 any\* parameter  
)

boolean setParameter(  
 int instance\_id,  
 ExtParameters ext\_parameters,  
 any\* parameter  
)

##### Definition

The getParameter and setParameter functions are extended with the following configuration parameters:

* PARAM\_SUBFRAME\_OUTPUT: this parameter is used to indicate if the output of subframes is required, desired, or not allowed. If it is not allowed, only complete decoded frames will be passed to the buffer.
* PARAM\_METADATA\_CALLBACK this parameter is used to set a callback function for a specific metadata type. The list of supported metadata types is codec dependent and shall be defined for each codec independently.
* PARAM\_OUTPUT\_CROP: this parameter is used to indicate that only part of the decoded frame is desired at the output. The decoder instance may use this information to intelligently reduce its decoding processing by discarding units that do not fall in the cropped output region whenever possible.
* PARAM\_MAX\_OFFTIME\_JITTER: this parameter is used to signal the maximum amount of time in microseconds between consecutive executions of the decoder instance. This parameter is relevant whenever the underlying hardware component is shared among multiple decoder instance, which requires context switching between the different decoder instances.

## Examples of Video Decoding Engine instantiations

### OpenMAX™

#### Decoder Engine Control Interface

##### OMX\_Init and OMX\_Deinit

Each OpenMAX IL client needs to call this method as their first call into OpenMAX™ IL. This function initializes the OMX core engine prior to any usage of it. Once done, the engine needs to be released by calling the OMX\_Deinit function.

OMX defines a naming convention for the component names with the following format: OMX.<vendor\_name>.<vendor\_specified\_convention>. Once the instance is no longer needed, the OMX\_FreeHandle is called to free all related resources.

The function can be called multiple times with the same component name to create multiple instances of the component.

##### OMX\_GetHandle and OMX\_FreeHandle

The OMX\_GetHandle method is used to locate the requested component through its provided name. If the requested component is available, the OMX core engine will invoke the components methods to fill the component handle and setup the callbacks. The OpenMAX™ AL is the interface that will be used by the application to perform media playback and processing. However, the OpenMAX™ IL interface is the interface that provides direct access to video decoder components and their capabilities. That is why we focus on the OpenMAX™ IL interface for the purpose of understanding the missing features towards a flexible multi-video decoder platform and its interface for 6DoF applications.

##### OMX\_SetupTunnel and OMX\_TeardownTunnel

A Tunnel is used to connect the input and output ports of two connected components. The OMX\_SetupTunnel is used to establish a tunnel connecting an output port to the input port of another component. When creating the tunnel, the components negotiate a compatible input/output format for the connected ports. When no longer needed, the application calls the OMX\_TeardownTunnel to tear down the tunnel.

#### Decoder Instance Interface

##### Methods

The components communicate among each other and with the application through buffer exchange. For this purpose, the OMX\_AllocateBuffer, OMX\_UseBuffer, OMX\_FillThisBuffer, OMX\_EmptyThisBuffer, and OMX\_FreeBuffer are defined. These function calls are non-blocking.

A component asks a preceding component to fill an input buffer by calling the OMX\_FillThisBuffer method and asks a succeeding component to retrieve the content of an output port buffer by calling the OMX\_EmptyThisBuffer method. Only one buffer per tunnel is allowed and one of the two components acts a supplier of that buffer.

Figure 3 depicts an example of connected components and the buffer usage:

A screenshot of a map

Description automatically generated

Figure 3 - Example of connected components and the buffer usage

The OMX\_SetConfig method is used to configure a component by the application. The application passes a structure that contains the configuration parameters to the component. The configuration parameters are published by each component and are component specific.

##### Media Input and Output Interface

The port configuration is used to define the format of the data to be transferred on a component port. The buffer header contains a reference to the buffer **pBuffer**, an offset inside that buffer **nOffset**, and the length of that buffer **nFilledLen**. Multiple buffers can be used to pass data, which allows for more flexibility in the communication between components, i.e. more than one frame can be exchanged at a time. Figure 4 shows an example:

A screenshot of a cell phone

Description automatically generated

Figure 4 - Port configuration example

There is no requirement on frame alignment to buffer start. The application or preceding components provide frame alignment information as part of the buffer header using the OMX\_BUFFERFLAG\_ENDOFFRAME flag. It is also possible to signal sub-frame boundaries to identify NAL unit boundaries using the OMX\_BUFFERFLAG\_ENDOFSUBFRAME.

A timestamp is also provided by the buffer header for every buffer. The **nTimestamp** corresponds to the presentation timestamp of the first media sample that starts at the current buffer. If multiple samples are included in the current buffer, the start timestamp of the following samples is inferred from the **nTimestamp** and the sample duration. That information can then be propagated through the pipeline and may be passed to the application through the output buffer.

The following picture shows the format of the buffer header:

A screenshot of a cell phone

Description automatically generated

The following buffer flags are defined in OpenMAX™ IL:

A close up of a newspaper

Description automatically generated

#### Input/Output from/into GPU

OpenMAX™ IL introduces the possibility to use an EGL Image as an output buffer. An EGL Image is designed for sharing data between rendering-based EGL interfaces, such as OpenGL and the OpenMAX™ components. It is up to the component to implement the OMX\_UseEGLImage to link the output to an EGL Image instead of a traditional buffer.

### MSE

#### Overview

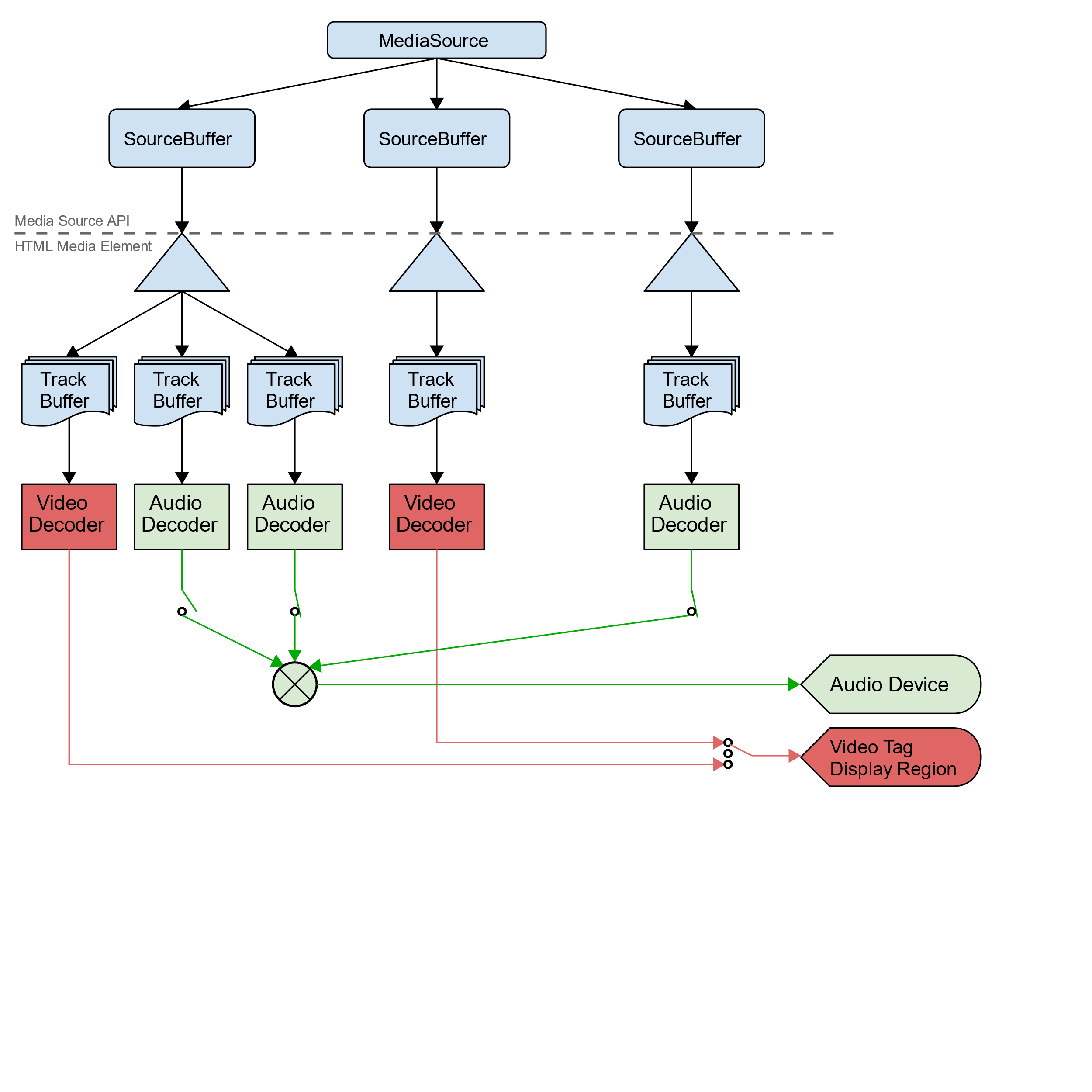


Figure 5 - Overview of MSE media interfaces

# Video decoder interface

As shown on Figure 1, the hardware video decoding engine may spawn one or more video decoder instances. The number of instances running is an optimisation choice for the platform when taking into account, computational load, energy consumption, memory availability etc. However, the number of input video streams fed through the IVDI is dictated by the application needs to properly render the media experience. As a result, one or more input video streams may be fed to the same video decoding instance as shown by the block called "Input formatting" in Figure 1.

The following section defines the binding for several video codecs to realise the operations on input video streams.

## Operations on input elementary streams

### Conventions

elementary streams

video object identifiers

### Concepts

ElementaryStream a type of elementary stream

AccessUnit a type of access unit

VideoObjectIdentifier a type of video object identifier

VideoObjectSample a type of video object sample

### General

The input formatting function in the Figure 1 provides several operations on elementary stream and video object. The input formatting function shall result in one or more elementary streams conforming to the profile, tier, level or any other performance constraints of the video decoder instance expected to consume it including buffer fullness of the hypothetical reference decoder model. These operations are defined in an atomic way such that more advanced operations can be achieved by combining the operations defined in this section. This list of operations is thus not an exhaustive list on purpose.

An elementary stream contains one or more video objects and a video object shall be contained in one elementary stream. Each video object in the elementary streams shall provide sufficient information for the operations such as a mean to determine the location and the size of the video object in the picture, the number of luma and chroma samples in the objects, the bit dept of the coded picture the video objects and so on.

**7.1.2 Filtering by video object identifier**

This function extracts one video object from an elementary stream and create an elementary stream that comprises the selected video object.

Function: Filtering

Definition:

Input: 1 elementary stream with multiple video objects

Identifier of the selected video object to be extracted

Output: 1 elementary stream with one video object which corresponds to the selected one

ElementaryStream output\_stream filtering(  
 ElementaryStream input\_stream,  
 ObjectIdentifier id) {

ElementaryStream new\_stream

for(au = begin(input\_stream); au != end(input\_stream); ++au) {

AccessUnit new\_au = au

for(ObjectSample object\_sample = begin(new\_au);  
 object\_sample!= end(new\_au);  
 ++object\_sample) {

if(identifier(object\_sample) != id) {

remove\_object(object\_sample, new\_au)

}

}

new\_stream << new\_au

}

return new\_stream

}

NOTE The extraction implements a filtering process based on the selected object identifier, that is the original access units are first copied and then removed from the unwanted objects. This way, the operation does not need to know how to create and initialize an empty access unit, but properties of the original access units are passed on to the access units of the output stream.

### Inserting video objects

This function inserts the video objects from a first elementary stream into a second elementary stream and output the resulting elementary stream that comprises the video objects from the first and second elementary streams.

Function: Inserting

Definition:

Input: 2 elementary streams with at least one video object each

Output: 1 elementary stream with as many video objects as the sum of video objects in both input elementary streams

ElementaryStream output\_stream inserting(  
 ElementaryStream input\_stream\_1,  
 ElementaryStream input\_stream\_2) {

ElementaryStream new\_stream

for(au\_1 = begin(input\_stream\_1), au\_2 = begin(input\_stream\_2);  
 au\_1 != end(input\_stream\_1), au\_2 != end(input\_stream\_2);  
 ++au\_1, ++au\_2) {

AccessUnit new\_au = au\_2

for(object\_sample = begin(au\_1);  
 object\_sample != end(au\_1);  
 ++object\_sample) {

add\_object(object\_sample, new\_au)

}

new\_stream << new\_au

}

return new\_stream

}

NOTE The inserting operation stops when one of the two input streams ends.

NOTE The inserting operation is defined as the insertion of video objects of the first elementary stream input into the second elementary stream input. This way, the operation does not need to know how to create and initialize an empty access unit, but the properties of the access units of the second elementary stream input are passed on to the access units of the output stream.

NOTE The function add\_object is defined for each video codec binding.

**7.1.4 Appending two video objects**

This function positions a first video object right of a second video object in the decoding pictures of the elementary streaming that contains those two video objects. The resulting elementary stream is an elementary containing at least the first and second video objects positioned as side-by-side neighbours.

Function: Appending

Definition:

Input: 1 elementary stream with at least two video objects

Output: 1 elementary stream with two video objects which are left and right spatial neighbours

ElementaryStream output\_stream appending(

ElementaryStream input\_stream,  
 ObjectIdentifier object\_id\_1,  
 ObjectIdentifier object\_id\_2) {

ElementaryStream new\_stream

for(au = begin(input\_stream); au != end(input\_stream); ++au) {

AccessUnit new\_au = au

set\_position(get\_object\_sample(new\_au, object\_id\_1),  
 right\_of(get\_object\_sample(new\_au, object\_id\_2)))

new\_stream << new\_au

}

return new\_stream

}

NOTE Appending is the operation of positioning video object 1 right of video object 0 with the top boundaries of video object 0 and video object 1 aligned.

NOTE The functions get\_object, right\_of, set\_position and get\_object\_sample are defined for each video codec binding.

**7.1.4 Stacking two video objects**

This function positions a first video object on top of a second video object in the decoding pictures of the elementary streaming that contains those two video objects. The resulting elementary stream is an elementary containing at least the first and second video objects positioned as top-and-bottom neighbours.

Function: Stacking

Definition:

Input: 1 elementary stream with at least two video objects

Output: 1 elementary stream with two video objects which are top and bottom spatial neighbours

ElementaryStream output\_stream stacking(

ElementaryStream input\_stream  
 ObjectIdentifier object\_id\_1,  
 ObjectIdentifier object\_id\_2) {

ElementaryStream new\_stream

for(au = begin(input\_stream); au != end(input\_stream); ++au) {

AccessUnit new\_au = au

set\_position(get\_object\_sample(new\_au, object\_id\_1),  
 below(get\_object\_sample(new\_au, object\_id\_2)))

new\_stream << new\_au

}

return new\_stream

}

NOTE Stacking is the operation of positioning video object 1 below video object 0 with the left boundaries of video object 0 and video object 1 aligned.

NOTE The functions get\_object, below and set\_position and get\_object\_sample are defined for each video codec binding.

## Instantiation for ISO/IEC 23090-2 Versatile Video Codec

### General

The Versatile Video Coding (VVC) is published under ISO/IEC 23090 part 3.

Table 1 provides the bindings of a concept of this document with a concept specified in the VVC specification.

Table 1 - Correspondence between concepts and VVC concrete entities

|  |  |
| --- | --- |
| Concept | VVC concept (reference) |
| ElementaryStream | bitstream (3.15) |
| AccessUnit | access unit (3.1) |
| VideoObjectIdentifier | nuh\_layer\_id (7.4.2.2) |
| VideoObjectSample | picture unit (3.19) |

**7.2.2 Elementary stream constraints**

[Editor's note] To be added.

## Instantiation for HEVC

### General

The High Efficiency Video Coding (HEVC) is published under ISO/IEC 23008 part 3.

Table 2 provides the bindings of a concept of this document with a concept specified in the HEVC specification.

Table 2 - Correspondence between concepts and VVC concrete entities

|  |  |
| --- | --- |
| Concept | HEVC concept (reference) |
| ElementaryStream | bitstream (3.15) |
| AccessUnit | access unit (3.1) |
| ObjectIdentifier | slice segment address (7.4.7.1) |
| ObjectSample | slice segment (3.151) |

**7.2.2 Elementary streams constraints**

[Editor's note] To be added

1. (normative)  
     
   Control Interface IDL Syntax
   1. General

The control interface to the video decoding platform is defined using the IDL syntax specified in ISO/IEC 19516 Information technology — Object management group — Interface definition language (IDL) 4.2.

* 1. Interface definition

interface OpenMAXILExt : OpenMAXIL {

const int CAP\_INSTANCES\_FLAG = 0x1;  
 const int CAP\_BUFFER\_MEMORY\_FLAG = 0x2;  
 const int CAP\_BITRATE\_FLAG = 0x4;  
 const int CAP\_MAX\_SAMPLES\_SECOND\_FLAG = 0x8;  
 const int CAP\_MAX\_PERFORMANCE\_POINT\_FLAG = 0xA;

enum ConfigParameters {  
 CONFIG\_OUTPUT\_BUFFER  
 };

enum ExtParameters {  
 PARAM\_SUBFRAME\_OUTPUT=1001,  
 PARAM\_METADATA\_CALLBACK,  
 PARAM\_OUTPUT\_CROP,  
 PARAM\_MAX\_OFFTIME\_JITTER  
 };

struct ConfigDataParameters {  
 sample\_format;  
 sample\_type;  
 sample\_stride;  
 line\_stride;  
 buffer\_offset  
 };

Editor's note: The above structure is missing the member types.

struct PerformancePoint {  
 float picture\_rate;  
 int width;  
 int height;  
 int bit\_depth;  
 };

struct AggregateCapabilities {  
 int flags;  
 int max\_instances;  
 int buffer\_memory;  
 int bitrate;  
 int max\_samples\_second;  
 PerformancePoint max\_performance\_point;  
 };

exception AllocationError {  
 string reason;  
 };

exception ConfigError {  
 string reason;  
 };

exception ParameterError {  
 string reason;  
 };

AggregateCapabilities queryCurrentAggregateCapabilities(  
 string component\_name,  
 int flags  
 );

int getInstance(  
 string component\_name,  
 int group\_id=-1  
 ) raises (AllocationError);

boolean setConfig(  
 int instance\_id,  
 ConfigParameters config\_parameters,  
 ConfigDataParameters config\_data\_parameters  
 ) raises (ConfigError);

any getParameter(  
 int instance\_id,  
 ExtParameters ext\_parameters,  
 any\* parameter  
 );

boolean setParameter(  
 int instance\_id,  
 ExtParameters ext\_parameters,  
 any\* parameter  
 ) raises (ParameterError);

}

Bibliography

1. ISO/IEC CD 23090-3, Information technology — Coded representation of immersive media — Part 3: Versatile video coding, "Text of ISO/IEC CD 23090-3 Versatile Video Coding", M18692
2. ISO/IEC JTC 1/SC 29/WG 11, "Requirements for Immersive Media Access and Delivery", N18654, Gothenburg, SE, July 2019