**ISO 23090-13:202x(E)**

ISO TC 1 SC 29/WG 03

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 v](#_Toc72927199)

[Introduction vi](#_Toc72927200)

[1 Scope 1](#_Toc72927201)

[2 Normative references 1](#_Toc72927202)

[3 Terms and definitions 1](#_Toc72927203)

[4 Timeline 2](#_Toc72927204)

[5 Motivations 2](#_Toc72927205)

[6 Video Decoding Engine 2](#_Toc72927206)

[6.1 General 2](#_Toc72927207)

[6.2 Input Video Decoding Interface 4](#_Toc72927208)

[6.2.1 Elementary stream 4](#_Toc72927209)

[6.2.2 Metadata stream 4](#_Toc72927210)

[6.3 Output Video Decoding Interface 4](#_Toc72927214)

[6.3.1 Decoded video sequences 5](#_Toc72927215)

[6.3.2 Metadata stream 5](#_Toc72927216)

[6.4 Control interface to the Video Decoding Interface 5](#_Toc72927221)

[6.4.1 Functions 5](#_Toc72927222)

[6.5 Examples of Video Decoding Engine instantiations 7](#_Toc72927223)

[6.5.1 Mapping on OpenMAX™ Integration Layer (OpenMAX IL) 7](#_Toc72927224)

[6.5.2 Mapping on Media Source Extensions (MSE) 8](#_Toc72927256)

[7 Video decoder interface 9](#_Toc72927257)

[7.1 Operations on input elementary streams 9](#_Toc72927258)

[7.1.1 General 9](#_Toc72927259)

[7.1.2 Conventions 10](#_Toc72927261)

[7.1.3 Concepts 10](#_Toc72927262)

[7.1.4 Inserting video objects 11](#_Toc72927263)

[7.1.5 Appending two video objects 13](#_Toc72927264)

[7.1.6 Stacking two video objects 14](#_Toc72927265)

[7.2 Instantiation for ISO/IEC 23008 part 3 HEVC 15](#_Toc72927266)

[7.2.1 General 15](#_Toc72927267)

[7.2.2 Elementary streams constraints 15](#_Toc72927268)

[7.3 Instantiation for ISO/IEC 23090-2 Versatile Video Codec 17](#_Toc72927269)

[7.3.1 General 17](#_Toc72927270)

[7.4 Instantiation for ISO/IEC 23094 part 1 EVC 19](#_Toc72927271)

[7.4.1 General 19](#_Toc72927272)

[7.4.2 Elementary streams constraints 20](#_Toc72927273)

[Annex A (normative) Control Interface IDL Syntax 22](#_Toc72927275)

[A.1 General 22](#_Toc72927276)

[A.2 Interface definition 22](#_Toc72927277)

[Annex B (normative) Supplemental Enhancement Information (SEI) syntax and semantics 24](#_Toc72927278)

[Annex C (informative) Brief description of OpenMAX IL functions 27](#_Toc72927279)

[C.1 Decoder Engine Control Interface 27](#_Toc72927280)

[C.1.1 OMX\_Init and OMX\_Deinit 27](#_Toc72927281)

[C.1.2 OMX\_GetHandle and OMX\_FreeHandle 27](#_Toc72927282)

[C.1.3 OMX\_SetupTunnel and OMX\_TeardownTunnel 27](#_Toc72927283)

[C.2 Decoder Instance Interface 27](#_Toc72927284)

[C.2.1 Methods 27](#_Toc72927285)

[C.2.2 Media Input and Output Interface 28](#_Toc72927286)

[C.2.3 Input/Output from/into GPU 29](#_Toc72927287)

[Bibliography 30](#_Toc72927288)

Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.

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.

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

elementary stream

stream of data composed of a sequence of access units

[Editor's note] There is need to either redefine an ES in VDI or to create a new term. The definition in MPEG-4 part 1 may no longer indeed apply as an “ES” in VDI may be decoded by multiple decoder instances after split by the input formatting which contradicts the original definition of an ES “consecutive flow of mono-media data from a single source entity to a single destination entity on the compression layer”)

3.2

metadata stream

TBD

[Editor's note] At MPEG #134, it has been identified that definition of ‘metadata stream’ is needed. One good example would be metadata track for V3C content. However, there is no ‘stream’ concept for metadata. It is simply serious of ISOBMFF file samples.

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:

* CD in July 2021
* FDIS April 2022

# Motivations

[Editor's note] Will be removed for CD stage.

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 the 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.

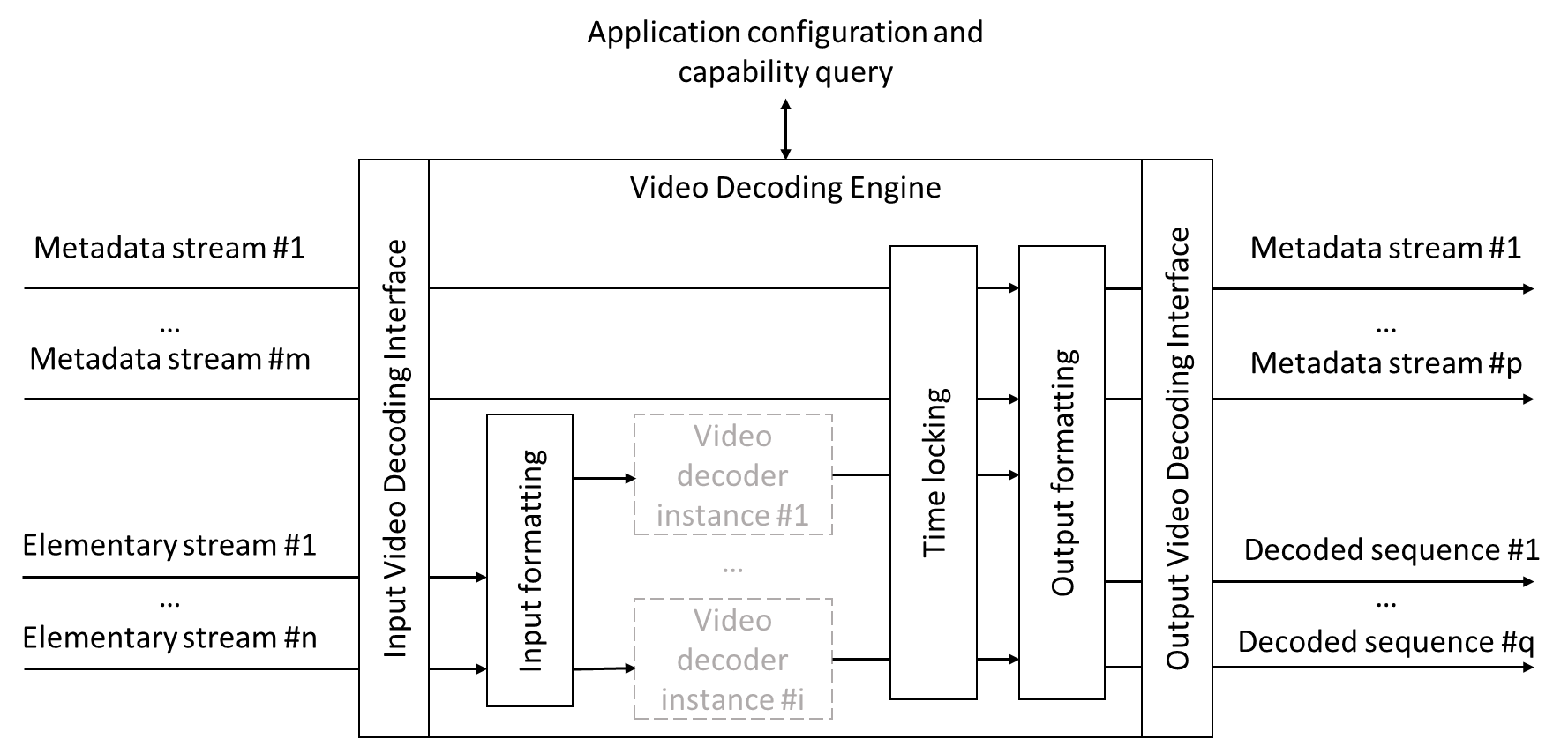


Figure 1 - Video Decoding Engine and interfaces

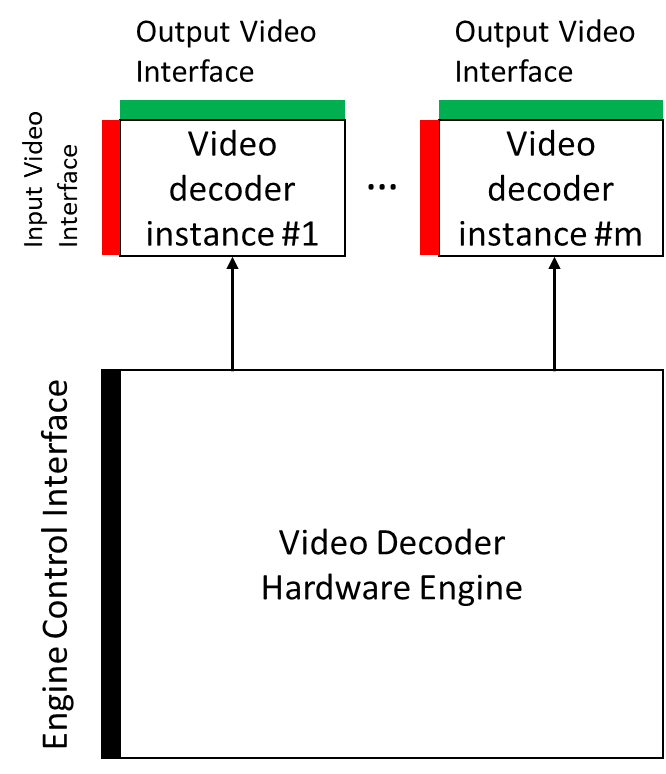


Figure 2 - Relation between video decoder instances and video decoder hardware engine

## Input Video Decoding Interface

The Video Decoding Engine accepts several data stream as input. Those data stream can be elementary streams or metadata streams. There is at least one elementary stream as input but there is no constraint on the number of metadata streams with respect to the number of elementary streams being concurrently consumed by the VDE.

### Elementary stream

[Editor's note] To be defined

### Metadata stream

[Editor's note] To be defined

## Output Video Decoding Interface

The Video Decoding Engine output two types of data stream. Those data streams can be decoded video sequences or metadata streams. There is at least one decoded video sequence as output but there is no constraint on the number of metadata streams with respect to the number of decoded video sequences being concurrently output by the VDE.

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

### Decoded video sequences

[Editor's note] To be defined

### Metadata stream

[Editor's note] To be defined. Is this the same as 6.2.2? Control interface to the Video Decoding Interface

## Control interface to the Video Decoding Interface

### Functions

In order to support immersive media applications, an abstract video decoding interface is specified. A video decoding platform that complies with this specification shall implement this video decoding interface.

The video decoding interface consists of the following abstract functions, which are defined using the IDL syntax specified in ISO/IEC 19516 Information technology — Object management group — Interface definition language (IDL) 4.2.

The following figure depicts example instantiation of decoder instances using some of the functionality of the video decoding interface.

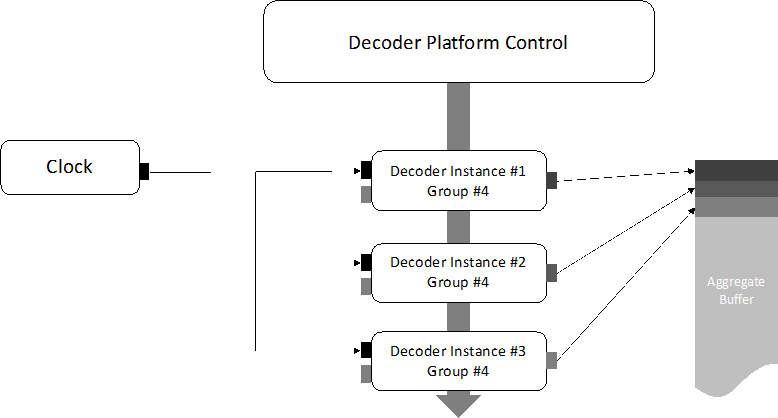


Figure 3 - Example Instantiation using VDI

#### 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\_SUBPICTURE\_OUTPUT: this parameter is used to indicate which sub-picture should be output by the decoder.
* 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

### Mapping on OpenMAX™ Integration Layer (OpenMAX IL)

#### Overview

The OpenMAX™ IL version 1.1.2 [3] is the baseline used for the design of section 6.4. For more information on OpenMAX IL, Annex C provides a short description of the main functions.

#### Mapping of VDI functions

The function definitions provided this section 6.4 map on OpenMAX IL interface by extending the existing functions for the ones with the same name or by creating new functions for the functions that don’t exist.

### Mapping on Media Source Extensions (MSE)

#### Overview

MSE is a set of extensions to the media source attributes of HTML5 video and audio elements. It enables flexible control of media streams through JavaScript through the definition of MediaSource objects. A MediaSource object may have one or more SourceBuffer objects. Applications append media segments to the SourceBuffer objects. A SourceBuffer may have multiple tracks, which are decoded and played separately. Figure 2 shows an example setup of MSE, depicting the interface between the MSE API and the HTML5 media element.

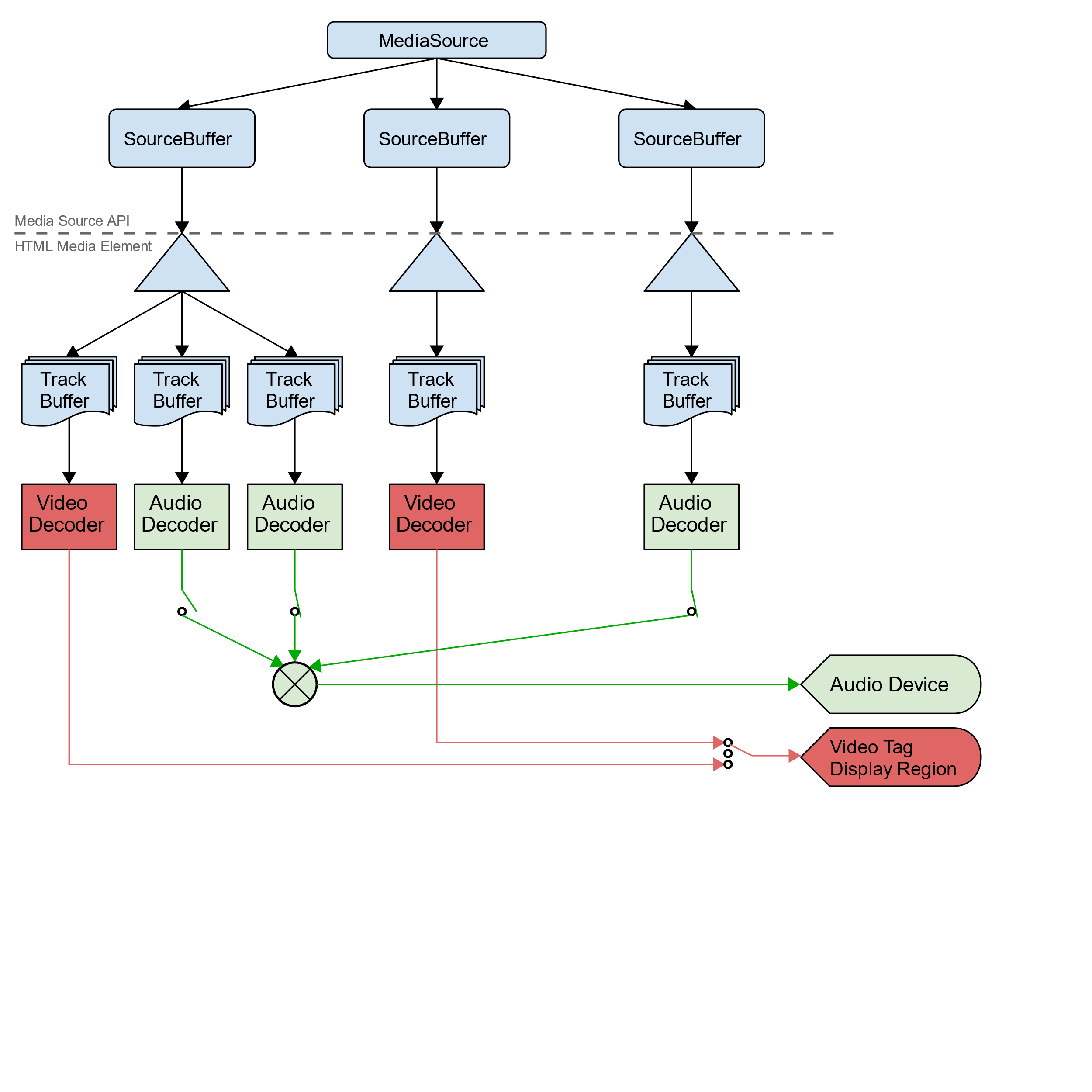


Figure 4 - Overview of MSE media interfaces

#### Mapping of VDI functions

The following table shows the potential mapping of the VDI functions to the MSE API.

|  |  |  |
| --- | --- | --- |
| VDI Functionality | MSE Mapping | Rationale |
| **queryCurrent AggregateCapabilities** | MediaSource.queryCurrent AggregateCapabilities | A new method of the MediaSource object is used to query the current decode capabilities. |
| **getInstance** with grouping | MediaSource.addSourceBuffer | With the current realization of MSE, tracks of the same type, e.g. VideoTracks, that belong to the same SourceBuffer are considered alternatives and only one is decoded and presented.  When creating a new SourceBuffer, a group identifier for each track type may be provided. This grouping applies across all currently instantiated MediaSource objects. This allows for grouping of multiple decoder instances that belong to multiple HTML5 media elements. |
| **setConfig**  CONFIG\_OUTPUT\_BUFFER | VideoTrack.setConfig | New method of the HTML5 VideoTrack object. |
| **getParameter** and **setParameter** | VideoTrack and AudioTrack getParameter and setParameter | New methods of HTML5 VideoTrack and AudioTrack objects. |

An extension to the HTML5 video element is recommended to allow for outputting data into buffers, e.g. WebGL buffers that created through gl.createBuffer() functions.

An extension to the input byte stream formats is recommended to add support for raw media data, e.g. H.264/AVC raw media streams.

# 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

### 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.

### 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

**7.1.2 Filtering by video object identifier**

#### Definition

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.

#### Example (informative)

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

For the filtering function, one slice is extracted from the input elementary stream and resulted as output elementary stream. For example, if the value of the smallest value of the ID of the tiles in the slice located in the top left corner is signalled as the argument of the filtering function, then such slice is extracted from the input elementary stream and the output elementary stream from the input formatting function will only contains that slice. Figure 5 (a) shows the video of input elementary stream and Figure 5 (b) shows video of the output elementary stream. During this operation, SPS, PPS and slice header needs to be rewritten to represent the size of the video of the output elementary stream, the information about the slices and tiles such as the number of tile columns, tile rows and the ID of the tiles needs to be updated.



Figure 5 - Input and output video examples of filtering function

### Inserting video objects

#### Definition

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.

#### Example (informative)

The inserting 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.

For the inserting function, the size of either width or height are identical for two videos used to maintain the shape of the video of the output elementary stream as rectangular shape. If the width of two videos are identical as shown in Figure 6 (a) and Figure 6 (b), then two videos are vertically concatenated as shown in Figure 6 (c). During this operation, SPS, PPS and slice header needs to be rewritten to represent the size of the video of the output elementary stream, the information about the slices and tiles such as the number of tile rows and the ID of the tiles needs to be updated. If the height of two videos are identical as shown in Figure 7 (a) and Figure 7 (b), then two videos are horizontally concatenated as shown in Figure 7 (c). During this operation, SPS, PPS and slice header needs to be rewritten to represent the size of the video of the output elementary stream, the information about the slices and tiles such as the number of tile columns and the ID of the tiles needs to be updated.



Figure 6 - Input and output video examples of inserting function for identical width



Figure 7 - Input and output video examples of inserting function for identical height

### Appending two video objects

#### Definition

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.

#### Example (informative)

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.

For the appending function, the height of the dotted slices in the input elementary stream to be used as the target of the operation in the video are identical as shown in Figure 8 (a). The dotted slice on the right side is moved to next to the dotted slice on the left side in the output elementary stream. The slice in between two slices are moved to right direction next to the two slices used as operation as shown in Figure 8 (b). During this operation, slice header needs to be rewritten to represent changes of the identifier of the slice as the identifiers of the tiles are modified.



Figure 8 - Input and output video examples of appending function

### Stacking two video objects

#### Definition

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.

#### Example (informative)

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.

For the stacking function, the width of the dotted slices in the input elementary stream to be used as the target of the operation in the video are identical as shown in Figure 9 (a). The dotted slice on the right side is moved to bottom of the dotted slice on the left side in the output elementary stream. The slice at the bottom of the dotted slide on the left side and the one right next to it are moved to the right direction sequentially as shown in Figure 9 (b). During this operation, slice header needs to be rewritten to represent changes of the identifier of the slice as the identifiers of the tiles are modified.



Figure 9 - Input and output video examples of stacking function

## Instantiation for ISO/IEC 23008 part 3 HEVC

### General

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

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

Table 1 - Correspondence between concepts and VVC concrete entities

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

### Elementary streams constraints

This section defines constraints to be applied to the input elementary streams of EVC to be used by Input Formatting function.

**7.2.3 General elementary stream constraints**

An EVC elementary stream used as an instantiation of the elementary stream in 7.1 Operations on input elementary streams shall obey the following rules:

* There shall be at least two independently decodable slices whose smallest value of the ID of the tiles in each slice are different are defined.

**7.2.4 Elementary stream constraints for input formatting functions**

**7.2.4.1 Constraints for the filtering function**

An EVC input elementary stream passed as argument of the filtering function shall comply to the following rules:

* One of the smallest values of the ID of the tiles in each slice shall be equal to the object identifier passed as argument of the filtering function.

An EVC elementary stream generated as output of the filtering function shall comply to the following rules:

* The number of access units in the output elementary stream shall be equal to the number of access units in the input elementary stream.
* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units with the smallest value of the ID of the tiles in the slice equal to object identifier passed as argument of the function.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.
* All the NAL units in the output elementary stream shall have the same smallest value of the ID of the tiles in the slice value and such value shall be equal to the object identifier passed as argument of the function.

**7.2.4.2 Constraints for the inserting function**

Two EVC input elementary streams passed as argument of the inserting function shall comply to the following rules:

* Al least one of the values of pic\_width\_in\_luma\_samples or pic\_height\_in\_luma\_samples of the two elementary streams shall be identical.
* If the values of pic\_width\_in\_luma\_samples are identical, then the values of num\_tile\_columns\_minus1 shall be identical.
* If the values of pic\_height\_in\_luma\_samples are identical, then the values of num\_tiles\_row\_minus1 shall be identical.
* If a SPS or PPS in the first input elementary stream has the same identifier than a SPS or PPS in the second input elementary stream, then those two SPSs or two PPSs shall have the same payload.

A EVC elementary stream generated as output of the inserting function shall comply to the following rules:

* The number of VCL NAL units in the output elementary stream is equal to the sum of the number of VCL NAL units in both input elementary streams.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in of one of the two input elementary streams that is bit exact identical.

**7.2.4.3 Constraints for the appending function**

An EVC input elementary stream passed as argument of the appending function shall comply to the following rules:

* At least two of the smallest values of the ID of the tiles in each slice shall be equal to the two object identifiers passed as arguments of the appending function.
* The height of the slices, number of tile rows of the tiles included in the slices when the uniform tile spacing is used, whose smallest values of the ID of the tiles in each slice are identical as arguments of the appending function are identical.

An EVC elementary stream generated as output of the appending function shall comply to the following rules:

* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units in the input elementary stream.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.

**7.2.4.4 Constraints for the stacking function**

An EVC input elementary stream passed as argument of the stacking function shall comply to the following rules:

* At least two of the smallest values of the ID of the tiles in each slice shall be equal to the two object identifiers passed as arguments of the appending function.
* The width of the slices, number of tile columns of the tiles included in the slices when the uniform tile spacing is used, whose smallest values of the ID of the tiles in each slice are identical as arguments of the appending function are identical.

An EVC elementary stream generated as output of the stacking function shall comply to the following rules:

* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units in the input elementary stream.

For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.

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

### General

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

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

Table 2 - 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) |

**7.2.3 General elementary stream constraints**

A VVC elementary stream used as an instantiation of the elementary stream in 7.1 Operations on input elementary streams shall obey the following rules:

* There shall be at least one VPS in the elementary stream and the parameters in each VPS shall be as follows:
  + The flag vps\_all\_independent\_layers\_flag shall be set to 1.
* The value of sh\_picture\_header\_in\_slice\_header\_flag shall be equal to 0 for all coded slices.
* When present, the value of vps\_num\_output\_layer\_sets\_minus2 shall be equal to the 0.

**7.2.4 Elementary stream constraints for input formatting functions**

**7.2.4.1 Constraints for the filtering function**

A VVC input elementary stream passed as argument of the filtering function shall comply to these rules in addition to the rules in 7.2.2:

* There shall be VCL NAL units with at least two different nuh\_layer\_id values.
* One of the at least two different nuh\_layer\_id values shall be equal to the object identifier passed as argument of the filtering function.

A VVC elementary stream generated as output of the filtering function shall comply to these rules in addition to the rules in 7.2.2:

* The number of access units in the output elementary stream shall be equal to the number of access units in the input elementary stream.
* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units with nuh\_layer\_id equal to object identifier passed as argument of the function.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.
* All the NAL units in the output elementary stream shall have the same nuh\_layer\_id value and this nuh\_layer\_id value shall be equal to the object identifier passed as argument of the function.

**7.2.4.2 Constraints for the inserting function**

Two VVC input elementary streams passed as argument of the inserting function shall comply to these rules in addition to the rules in 7.2.2:

* The nuh\_layer\_id value of each NAL unit in the first input elementary stream shall be different from any nuh\_layer\_id value present in the second input elementary stream.
* If a SPS or PPS in the first input elementary stream has the same identifier than a SPS or PPS in the second input elementary stream, then those two SPSs or two PPSs shall have the same payload.

A VVC elementary stream generated as output of the inserting function shall comply to these rules in addition to the rules in 7.2.2:

* The number of VCL NAL units in the output elementary stream is equal to the sum of the number of VCL NAL units in both input elementary streams.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in of one of the two input elementary streams that is bit exact identical.

**7.2.4.3 Constraints for the appending function**

A VVC input elementary stream passed as argument of the appending function shall comply to these rules in addition to the rules in 7.2.2:

* There shall be VCL NAL units with at least two different nuh\_layer\_id values.
* Two of the at least two different nuh\_layer\_id values shall be equal to the two object identifiers passed as arguments of the appending function.

A VVC elementary stream generated as output of the appending function shall comply to these rules in addition to the rules in 7.2.2:

* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units in the input elementary stream.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.
* There shall be an Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the first video object identifier.
* There shall be an Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the second video object identifier.
* The Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the first video object identifier shall have its boundary\_identifier\_east value equal to the boundary\_identifier\_west value of the Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the second video object identifier.

**7.2.4.4 Constraints for the stacking function**

A VVC input elementary stream passed as argument of the stacking function shall comply to these rules in addition to the rules in 7.2.2:

* There shall be VCL NAL units with at least two different nuh\_layer\_id values.
* Two of the at least two different nuh\_layer\_id values shall be equal to the two object identifiers passed as arguments of the appending function.

A VVC elementary stream generated as output of the stacking function shall comply to these rules in addition to the rules in 7.2.2:

* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units in the input elementary stream.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.
* There shall be an Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the first video object identifier.
* There shall be an Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the second video object identifier.
* The Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the first video object identifier shall have its boundary\_identifier\_south value equal to the boundary\_identifier\_north value of the Independent layer info SEI NAL unit whose nuh\_layer\_id is equal to the second video object identifier.

## Instantiation for ISO/IEC 23094 part 1 EVC

### General

The Essential Video Coding (EVC) is published under ISO/IEC 23094 part 1.

Table 13 provides the bindings of a concept of this document with a concept specified in the EVC specification.

Table 3 - Correspondence between concepts and EVC concrete entities

|  |  |
| --- | --- |
| Concept | EVC concept (reference) |
| ElementaryStream | bitstream (3.10) |
| AccessUnit | access unit (3.1) |
| ObjectIdentifier | the smallest value of the ID of the tiles in a slice (7.4.5) |
| ObjectSample | slice (3.81) |

### Elementary streams constraints

#### General elementary stream constraints

An EVC elementary stream used as an instantiation of the elementary stream in 7.1 Operations on input elementary streams shall obey the following rules:

* There shall be at least two independently decodable slices whose smallest value of the ID of the tiles in each slice are different are defined

**7.2.4 Elementary stream constraints for input formatting functions**

**7.2.4.1 Constraints for the filtering function**

An EVC input elementary stream passed as argument of the filtering function shall comply to the following rules:

* One of the smallest values of the ID of the tiles in each slice shall be equal to the object identifier passed as argument of the filtering function.

An EVC elementary stream generated as output of the filtering function shall comply to the following rules:

* The number of access units in the output elementary stream shall be equal to the number of access units in the input elementary stream.
* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units with the smallest value of the ID of the tiles in the slice equal to object identifier passed as argument of the function.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.
* All the NAL units in the output elementary stream shall have the same smallest value of the ID of the tiles in the slice value and such value shall be equal to the object identifier passed as argument of the function.

**7.2.4.2 Constraints for the inserting function**

Two EVC input elementary streams passed as argument of the inserting function shall comply to the following rules:

* Al least one of the values of pic\_width\_in\_luma\_samples or pic\_height\_in\_luma\_samples of the two elementary streams shall be identical.
* If the values of pic\_width\_in\_luma\_samples are identical, then the values of num\_tile\_columns\_minus1 shall be identical.
* If the values of pic\_height\_in\_luma\_samples are identical, then the values of num\_tiles\_row\_minus1 shall be identical.
* If a SPS or PPS in the first input elementary stream has the same identifier than a SPS or PPS in the second input elementary stream, then those two SPSs or two PPSs shall have the same payload.

A EVC elementary stream generated as output of the inserting function shall comply to the following rules:

* The number of VCL NAL units in the output elementary stream is equal to the sum of the number of VCL NAL units in both input elementary streams.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in of one of the two input elementary streams that is bit exact identical.

**7.2.4.3 Constraints for the appending function**

An EVC input elementary stream passed as argument of the appending function shall comply to the following rules:

* At least two of the smallest values of the ID of the tiles in each slice shall be equal to the two object identifiers passed as arguments of the appending function.
* The height of the slices, number of tile rows of the tiles included in the slices when the uniform tile spacing is used, whose smallest values of the ID of the tiles in each slice are identical as arguments of the appending function are identical.

An EVC elementary stream generated as output of the appending function shall comply to the following rules:

* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units in the input elementary stream.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.

**7.2.4.4 Constraints for the stacking function**

An EVC input elementary stream passed as argument of the stacking function shall comply to the following rules:

* At least two of the smallest values of the ID of the tiles in each slice shall be equal to the two object identifiers passed as arguments of the appending function.
* The width of the slices, number of tile columns of the tiles included in the slices when the uniform tile spacing is used, whose smallest values of the ID of the tiles in each slice are identical as arguments of the appending function are identical.

An EVC elementary stream generated as output of the stacking function shall comply to the following rules:

* The number of VCL NAL units in the output elementary stream is equal to the number of VCL NAL units in the input elementary stream.
* For each VCL NAL unit in the output elementary stream, there shall exist a VCL NAL unit in the input elementary stream that is bit exact identical.

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);

}

1. (normative)  
     
   Supplemental Enhancement Information (SEI) syntax and semantics
   1. **Independent layer info SEI message**
      1. **Independent layer info SEI message syntax**

Table 4 - Syntax of independent layer info SEI payload

|  |  |
| --- | --- |
| **Syntax** | **Descriptor** |
| independent\_layer\_info( payloadSize ) { |  |
| **boundary\_identifier\_north\_present\_flag** | u(1) |
| if(boundary\_identifier\_north\_present\_flag) |  |
| **boundary\_identifier\_north** | u(16) |
| **boundary\_identifier\_east\_present\_flag** | u(1) |
| if(boundary\_identifier\_east\_present\_flag) |  |
| **boundary\_identifier\_east** | u(16) |
| **boundary\_identifier\_south\_present\_flag** | u(1) |
| if(boundary\_identifier\_south\_present\_flag) |  |
| **boundary\_identifier\_south** | u(16) |
| **boundary\_identifier\_west\_present\_flag** | u(1) |
| if(boundary\_identifier\_west\_present\_flag) |  |
| **boundary\_identifier\_west** | u(16) |
| } |  |

* + 1. **Independent layer info SEI message semantics**

**boundary\_identifier\_north\_present\_flag, boundary\_identifier\_east\_present\_flag, boundary\_identifier\_south\_present\_flag** and **boundary\_identifier\_west\_present\_flag** equal to 1 specify that the SEI message contains a boundary identifier, respectively, for the north, east, south and west boundary.

**boundary\_identifier\_north**, **boundary\_identifier\_east**, **boundary\_identifier\_south** and **boundary\_identifier\_west** specifiy the boundary identifier, respecitvely, at the north, east, south and west boundary of the decoded picture of the associated layer; the associated layer being the layer whose nuh\_layer\_id is equal to the nuh\_layer\_id of the SEI message. If not present, the boundary identifiers respecitvely at the north, east, south and west boundary of the decoded picture of the associated layer are not defined.

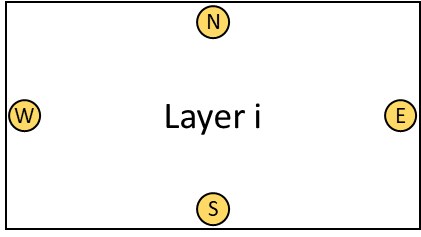


Figure 10 - Representation of where the boundaries are on the layers

For two layers, the i-th and j-th layers, when the pair of the boundary\_identifier\_north value of the i-th layer and the boundary\_identifier\_south value of the j-th layer are equal then the decoded picture of the i-th layer and the decoded picture of the j-th layer are adjacent in the composed output picture and they share a common boundary at the boundary north/south. For i-th and j-th layers, when the pair of the boundary\_identifier\_east value of the i-th layer and the boundary\_identifier\_west value of the j-th layer are equal then the decoded picture of the i-th layer and the decoded picture of the j-th layer are adjacent in the composed output picture and they share a common boundary at the boundary east/west.

Two decoded pictures adjacent by the north/south boundary are aligned on their west boundary in the final output picture. Two decoded pictures adjacent by the east/west boundary are aligned on their north boundary in the final output picture.

All the Independent layer info SEI message present in layers of an OLS shall collectively describe a 4-connected graph and each layer of the OLS shall be connected to the graph.

* + 1. **Process for generating the aggregated output picture**

The process for generating the final output picture is informative. The following section provides the expected operations performed for generating the final output picture based on the decoded pictures of each layer from a selected OLS:

* For each Access Unit:
  + If VPS present, parse VPS and store the list of layers in the bitstream.
  + For each present PPS, determine the size in luma samples of the corresponding layer.
  + For each present Independent layer info SEI, parse the payload and store the boundary identifiers for the corresponding layer.
  + If any of VPS, PPS or Independent layer info SEI is present in the current Access Unit, execute calculate\_layer\_layout\_in\_final\_output\_picture(), that returns the horizontal, XPos, and vertical, YPos, position of the top-left corner of each cropped decoded picture per layer in the final output picture.
  + Initialize a picture buffer of size FinalWidth of width and FinalHeight of height for the final output picture.
  + For each Picture Unit:
    - Decode the coded picture.
  + If pictures are ready for output:
    - For each layer in selected OLS.
      * Apply conformance window cropping on the decoded picture of the current layer.
      * Retrieve XPos and YPos, the positions of the current layer in the final output picture in luma sample.
      * Copy cropped decoded picture in final output picture buffer at position (XPos, YPos) corresponding to the top-left corner of the cropped decoded picture.

1. (informative)  
     
   Brief description of OpenMAX IL functions
   1. Decoder Engine Control Interface
      1. 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.

* + 1. 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.

* + 1. 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.

* 1. Decoder Instance Interface
     1. 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.

depicts an example of connected components and the buffer usage:

A screenshot of a map

Description automatically generated

Figure 11 - 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.

* + 1. 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. shows an example:

A screenshot of a cell phone

Description automatically generated

Figure 12 - 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

* + 1. 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.

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
3. OpenMAX™ Integration Layer Application Programming Interface Specification Version 1.1.2, The Khronos Group Inc.