**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**

CD 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](#_Toc78533706)

[Introduction vi](#_Toc78533707)

[1 Scope 1](#_Toc78533708)

[2 Normative references 1](#_Toc78533709)

[3 Terms and definitions 1](#_Toc78533711)

[4 Video Decoding Engine 2](#_Toc78533716)

[4.1 General 2](#_Toc78533717)

[4.2 Input Video Decoding Interface 4](#_Toc78533718)

[4.3 Output Video Decoding Interface 4](#_Toc78533723)

[4.4 Control interface to the Video Decoding Interface 5](#_Toc78533728)

[4.4.1 Functions 5](#_Toc78533729)

[4.5 Examples of Video Decoding Engine instantiations 9](#_Toc78533730)

[4.5.1 Mapping on OpenMAX™ Integration Layer (OpenMAX IL) 9](#_Toc78533731)

[4.5.2 Mapping on Media Source Extensions (MSE) 9](#_Toc78533732)

[5 Video decoder interface 10](#_Toc78533733)

[5.1 Operations on input media streams 10](#_Toc78533734)

[5.1.1 General 10](#_Toc78533735)

[5.1.2 Conventions 11](#_Toc78533736)

[5.1.3 Concepts 11](#_Toc78533737)

[5.1.4 Filtering by video object identifier 11](#_Toc78533738)

[5.1.5 Inserting video objects 12](#_Toc78533739)

[5.1.6 Appending two video objects 14](#_Toc78533740)

[5.1.7 Stacking two video objects 15](#_Toc78533741)

[5.2 Instantiation for ISO/IEC 23008-3 High Efficiency Video Coding (HEVC) 16](#_Toc78533742)

[5.2.1 General 16](#_Toc78533743)

[5.3 Instantiation for ISO/IEC 23090-2 Versatile Video Codec (VVC) 16](#_Toc78533747)

[5.3.1 General 16](#_Toc78533748)

[5.3.2 Media and elementary stream constraints 17](#_Toc78533749)

[5.4 Instantiation for ISO/IEC 23094-1 Essential Video Coding (EVC) 18](#_Toc78533750)

[5.4.1 General 18](#_Toc78533751)

[5.4.2 Media and elementary streams constraints 19](#_Toc78533752)

[Annex A (normative) Control interface IDL definition 21](#_Toc78533753)

[A.1 General 21](#_Toc78533754)

[A.2 Interface definition 21](#_Toc78533755)

[Annex B (normative) OpenMAX IL VDI extension header 23](#_Toc78533756)

[B.1 General 23](#_Toc78533757)

[B.2 Source code 23](#_Toc78533758)

[Annex C (normative) Supplemental Enhancement Information (SEI) syntax and semantics 26](#_Toc78533759)

[Annex D (informative) Brief description of OpenMAX IL functions 29](#_Toc78533760)

[D.1 Decoder Engine Control Interface 29](#_Toc78533761)

[D.1.1 OMX\_Init() and OMX\_Deinit() 29](#_Toc78533762)

[D.1.2 OMX\_GetHandle() and OMX\_FreeHandle() 29](#_Toc78533763)

[D.1.3 OMX\_SetupTunnel() and OMX\_TeardownTunnel() 29](#_Toc78533764)

[D.2 Decoder Instance Interface 29](#_Toc78533765)

[D.2.1 Methods 29](#_Toc78533766)

[D.2.2 Media Input and Output Interface 30](#_Toc78533767)

[D.2.3 Input/Output from/into GPU 31](#_Toc78533768)

[Bibliography 32](#_Toc78533769)

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 for 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.

Those interfaces and operations come as extensions of existing video decoding engines exposing hardware video decoding capabilities.

# 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 19516, *Information technology — Object management group — Interface definition language (IDL) 4.2*

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 23094-1, *Information technology — General video coding — Part 1: Essential video coding*

# Terms and definitions

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

3.1

access unit

smallest individually accessible portion of data within an *elementary stream* (3.2) to which unique timing information can be attributed

[SOURCE: ISO/IEC 14496-1]

3.2

elementary stream

consecutive flow of mono-media data produced by a single encoder or consumed by a single decoder

NOTE 1 to entry: Similar to the definition of elementary stream in ISO/IEC 14496-1.

3.3

media stream

one or more aggregated *elementary streams* (3.2)

NOTE 1 to entry: Every elementary stream is a media stream but the inverse is not true.

3.4

video object

independently decodable substream of a video *elementary stream* (3.2)

NOTE 1 to entry: Examples of such data types are visual textures, depth maps, occupancy maps, etc.

3.5

video object identifier

an integer identifying a *video object* (3.4)

# Video Decoding Engine

## General

The Video Decoding Engine (VDE) enables the decoding, the synchronisation and the formatting of one or more aggregated elementary streams, called media 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) in their decoded form. The VDE extracts and merges independently decodable regions of a set of input media streams and generates a set of elementary streams fed to the video decoder instances, which run inside the engine, through the input formatting function. The VDE can execute a merging operation or an extraction operation on the input media streams when the number of available video decoder instances is different from the number of input media streams while the VDE has sufficient computational power and available resources to decode all the independently decodable regions from the input media streams that are required by the application. For example, a VDE may not be capable of decoding a input 4K elementary stream with one decoder instance but it may be able decode some of the independently decodable regions present in this input elementary stream.

Figure 1 represents the architecture for the VDE and the associated IVDI and OVDI interfaces.



**Key**

|  |  |  |  |
| --- | --- | --- | --- |
| MDS | media stream |  |  |
| ES | elementary stream |  |  |
| MTS | metadata stream |  |  |
| DS | decoded sequence |  |  |

NOTE Multiple elementary streams that are output of the input formatting function may be fed to a single video decoder instance.

Figure 1 — Video Decoding Engine and 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 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.

The input of the VDE comprises thus:

* n elementary streams
* m metadata streams

## 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.

The output of the VDE comprises thus:

* q decoded sequences
* p metadata streams

## 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.

Figure 3 depicts an example instantiation of decoder instances using some of the functionalities of the video decoding interface.



Figure 3 — Example Instantiation using VDI

#### queryCurrentAggregateCapabilities()

##### Declaration

Table 1 reproduces the IDL declarations of the queryCurrentAggregateCapabilities() operation along with the AggregateCapabilities and PerformancePoint structures and the capabilities flags.

Table 1 — IDL declarations of queryCurrentlAggregateCapabilities(), AggregateCapabilities, PerformancePoint and capabilities flags

|  |
| --- |
| const unsigned long CAP\_INSTANCES\_FLAG = 0x1;  const unsigned long CAP\_BUFFER\_MEMORY\_FLAG = 0x2;  const unsigned long CAP\_BITRATE\_FLAG = 0x4;  const unsigned long CAP\_MAX\_SAMPLES\_SECOND\_FLAG = 0x8;  const unsigned long CAP\_MAX\_PERFORMANCE\_POINT\_FLAG = 0xA;  struct PerformancePoint {  float picture\_rate;  unsigned long width;  unsigned long height;  unsigned long bit\_depth;  };  struct AggregateCapabilities {  unsigned long flags;  unsigned long max\_instances;  unsigned long buffer\_memory;  unsigned long bitrate;  unsigned long max\_samples\_second;  PerformancePoint max\_performance\_point;  };  AggregateCapabilities queryCurrentAggregateCapabilities (  in string component\_name,  in unsigned long flags  ); |

##### Definition

The queryCurrentAggregateCapabilities() 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 capability flags below can set separately or in a single function call to query one or more parameters.

CAP\_INSTANCES\_FLAG queries the max\_instances parameter which indicates the maximum number of decoder instances that can be instantiated at this instant for the provided decoder component.

CAP\_BUFFER\_MEMORY\_FLAG queries the buffer\_memory parameter which indicates the maximum available buffer size in bytes that can be allocated at this instant on the decoder platform for buffer exchange with components of the media decoding platform.

CAP\_BITRATE\_FLAG queries the bitrate parameter which indicates the maximum coded bitrate in bits per second that the queried component is able to process at this instant.

CAP\_MAX\_SAMPLES\_SECOND\_FLAG queries the max\_samples\_second parameter which indicates the maximum number of samples per second that the queried component is able to process at this instant.

CAP\_MAX\_PERFORMANCE\_POINT\_FLAG queries the max\_performance\_point parameter which indicates 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 contains the following parameters:

* picture\_rate indicating the picture rate of the maximum performance point in pictures per second.
* height indicating the height in luma samples of the maximum performance point
* width indicating the width in luma samples of the maximum performance point
* bit\_depth indicating the bit depth of the luma samples of the maximum performance point

NOTE CAP\_BUFFER\_MEMORY is independent from any media components and applies globally to the decoding platform.

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

Table 2 reproduces the IDL declarations of the getInstance() operation and the associated ErrorAllocation exception.

Table 2 — IDL declarations of getInstance() and ErrorAllocation

|  |
| --- |
| exception ErrorAllocation {  string reason;  };  unsigned long getInstance(  in string component\_name,  in unsigned long group\_id // optional, default value = -1  ) raises(ErrorAllocation); |

##### 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

Table 3 reproduces the IDL declarations of the setConfig() operation, the associated ErrorConfig exception, the ConfigDataParameters structure and the ConfigParameters enumeration.

Table 3 — IDL declarations of setConfig(), ErrorConfig, ConfigParameters and ConfigDataParameters

|  |
| --- |
| enum ConfigParameters {  CONFIG\_OUTPUT\_BUFFER  };  struct ConfigDataParameters {  SampleFormat sample\_format;  SampleType sample\_type;  unsigned long sample\_stride;  unsigned long line\_stride;  unsigned long buffer\_offset;  };  exception ErrorConfig {  string reason;  };  boolean setConfig (  in unsigned long instance\_id,  in ConfigParameters config\_parameters,  in ConfigDataParameters config\_data\_parameters  ) raises(ErrorConfig); |

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

#### getParemeter() and setParameter()

##### Declaration

Table 4 reproduces the IDL declarations of the getParameter() and setParemeter() operations as well as the associated ErrorParameter exception and the ExtParameters enumeration.

Table 4 — IDL declarations of getParameter(), setParameter() and ErrorParameter and ExtParameters

|  |
| --- |
| enum ExtParameters {  PARAM\_SUBFRAME\_OUTPUT,  PARAM\_METADATA\_CALLBACK,  PARAM\_OUTPUT\_CROP,  PARAM\_MAX\_OFFTIME\_JITTER  };  any getParameter (  in unsigned long instance\_id,  in ExtParameters ext\_parameters,  out any parameter  );  boolean setParameter (  in unsigned long instance\_id,  in ExtParameters ext\_parameters,  in any parameter  ) raises(ErrorParameter); |

##### Definition

The getParameter() and setParameter() functions can receive the extended parameters below.

PARAM\_SUBFRAME\_OUTPUT indicates whether 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 sets 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 indicates 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 indicates which sub-picture should be output by the decoder.

PARAM\_MAX\_OFFTIME\_JITTER indicates 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

For more information on OpenMAX IL, Annex D provides a short description of the main functions of this API.

#### Mapping of VDI functions

The function defined in Clause 4.4 are mapped on the OpenMAX IL interface by using extension mechanism defined by specification author. This MPEG VDI extension for OpenMAX IL is formatted as a C header file and registered with the vendor name “MPEG”.

[Editor’s note] MPEG prefix to be registered for OpenMAX IL.

Annex B reproduces the MPEG VDI extension for OpenMAX Il and provides information to access the electronic version of the extension.

### 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 code using 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 4 shows an example setup of MSE which depicts the interface between the MSE API and the HTML5 media element.

Track  
Buffer

Audio Device

Video Tag

Display Region

Track  
Buffer

Track  
Buffer

Track  
Buffer

Track  
Buffer

Video Decoder

Audio Decoder

Audio Decoder

Audio Decoder

Video Decoder

SourceBuffer

SourceBuffer

SourceBuffer

MediaSource

Media Source API

HTML Media Element

Figure 4 — Overview of MSE media interfaces

#### Mapping of VDI functions

Table 5 shows the possible mapping of the VDI functions onto the MSE API.

Table 5 — Possible mapping of VDI onto MSE

|  |  |
| --- | --- |
| **VDI Functionality** | **MSE Mapping** |
| queryCurrentAggregate Capabilities() | MediaSource.queryCurrentAggregate Capabilities() a |
| getInstance() with grouping | MediaSource.addSourceBuffer() b |
| setConfig() CONFIG\_OUTPUT\_BUFFER | VideoTrack.setConfig() c |
| getParameter() and setParameter() | VideoTrack and AudioTrack, getParameter() and setParameter() d |
| NOTE   Rationale and description are provided below.  a   A new method of the MediaSource object is used to query the current decode capabilities.  b   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.  c  New method of the HTML5 VideoTrack object.  d  New methods of HTML5 VideoTrack and AudioTrack objects. | |

In addition, 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. AVC raw media streams.

# Video decoder interface

As shown in 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.

Clause 5 defines the binding for several video codecs to realise the operations on input video streams.

## Operations on input media streams

### General

The input formatting function in Figure 1 provides several operations on media streams and video objects. 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 complex operations can be achieved by combining the operations defined in Clause 5.1. This list of operations is thus not an exhaustive list on purpose.

A media stream contains one or more video objects and a video object shall be contained into one elementary stream. Each video object in an elementary stream shall provide sufficient information for the defined operations such as a mean to determine the location and the dimension of the video object in the picture, the number of luma and chroma samples in the video object, the bit depth of the coded picture of the video object and so on.

### Conventions

media stream

elementary stream

video object identifier

### Concepts

MediaStream a type of media stream

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

### Filtering by video object identifier

#### Definition

Function: Filtering

Definition:

Input: 1 media stream with at least one video object

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(  
 MediaStream 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 function remove\_object() is defined for each video codec binding.

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 create and initialize an empty access unit, but properties of the input access units are passed on to the access units of the output stream.

#### Description

The filtering function extracts one video object from an input media stream and returns an elementary stream as output which comprises the selected video object.

In case the video object is a slice, the filtering function extracts this slice in every coded picture from the input media stream and passes it in the output elementary stream. This case is illustrated in Figure 5 wherein the diagram (a) shows the video of the input media stream and the diagram **Error! Reference source not found.**(b) shows the video of the output elementary stream. During this operation, the SPS, PPS and slice header may need to be updated as required by the corresponding video coding specification to correctly signal the size of the video of the output elementary stream, the information about the slices and tiles layout and the video object identifier, e.g. the slice address.



Figure 5 — Example of input and output video for the filtering function

### Inserting video objects

#### Definition

Function: Inserting

Definition:

Input: 2 media streams containing at least one video object each

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

MediaStream output\_stream inserting(  
 MediaStream input\_stream\_1,  
 MediaStream 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 as soon as one of the two input streams ends.

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

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

#### Description

The inserting function takes the video objects from a first input media stream into a second input media stream and output the resulting output media stream which comprises the video objects from both first and second input media streams.

In case the video objects are slices, either the width or the height of the coded pictures of the input media streams are equal in order to maintain the rectangular shape of the video of the output media stream. In case the width of the two input videos are equal, as shown in the diagrams (a) and (b) in Figure 6, then the two videos are vertically stitched as shown in the diagram (c) of the same figure. During this operation, the SPS, PPS and slice header may need to be updated to correctly signal the size of the video of the output media stream, the information about the slices and tiles layout and the video object identifiers, e.g. the slice addresses.



Figure 6 — Example of input and output video for the inserting function with identical width

If the height of two videos are equal, as shown in the diagrams (a) and (b) in Figure 7, then the two videos are horizontally stitched as shown in the diagram (c) of the same figure. During this operation as well, the SPS, PPS and slice header may need to be updated to correctly signal the size of the video of the output media stream, the information about the slices and tiles layout and the video object identifiers, e.g. the slice addresses.



Figure 7 — Example of input and output video for the inserting function with identical height

### Appending two video objects

#### Definition

Function: Appending

Definition:

Input: 1 media stream with at least two video objects

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

MediaStream output\_stream appending(

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

MediaStream 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 2 right of video object 1 with the top boundaries of video object 1 and video object 2 aligned.

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

#### Description

The appending function positions a first video object right of a second video object in the decoded pictures of the output media stream, which contains those two video objects. The output media stream is a media containing at least the first and second video objects positioned as side-by-side neighbours.

In case the video object is a slice, the slices of both video objects in the input media stream have the same height as shown in the example Figure 8, diagram (a). The slice on the right side is moved next to the slice on the left side in the output media stream. The slice in between the two slices is moved to the right next to the two slices used as input of the operation as shown in the diagram (b) of the same figure. During this operation, slice header may need to updated to correctly signal the changes of the video object identifiers, e.g. the slice addresses.



Figure 8 — Example of input and output video for the appending function

### Stacking two video objects

#### Definition

Function: Stacking

Definition:

Input: 1 media stream with at least two video objects

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

MediaStream output\_stream stacking(

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

MediaStream 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 2 below video object 1 with the left boundaries of video object 1 and video object 2 aligned.

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

#### Description

The stacking function positions a first video object on top of a second video object in the decoded pictures of the media stream that contains those two video objects. The output media stream contains at least the first and second video objects positioned as top-and-bottom neighbours.

In case the video object is a slice, the slices of both video objects in the input media stream have the same width as shown in the example Figure 9, diagram (a). The slice on the right side is moved to below the slice on the left side in the output media stream. The slice at the below the slice on the left side and the one right next to it are moved to the right direction sequentially as shown in the diagram (b) of the same figure. During this operation, the slice header may need to be updated to correctly signal the changes of the video object identifiers, e.g. the slice addresses.



Figure 9 — Example of input and output video for the stacking function

## Instantiation for ISO/IEC 23008-3 High Efficiency Video Coding (HEVC)

### General

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

Table 6 provides the bindings of a concept of this document with a concept specified in the HEVC specification.

Table 6 — Correspondence between VDI concepts and HEVC 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) |

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

### General

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

Table 7 provides the bindings of a concept of this document with a concept specified in the VVC specification.

Table 7 — Correspondence between VDI 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) |

### Media and elementary stream constraints

#### General media stream constraints

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

* There shall be at least one VPS in the media 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.

#### Media and elementary stream constraints for input formatting functions

##### Constraints for the filtering function

A VVC input media 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.

##### Constraints for the inserting function

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

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

##### Constraints for the appending function

A VVC input media 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 media 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 media stream is equal to the number of VCL NAL units in the input media stream.
* For each VCL NAL unit in the output media stream, there shall exist a VCL NAL unit in the input media stream that is bit exact identical.
* There shall be an Independent layer info SEI message whose nuh\_layer\_id is equal to the first video object identifier.
* There shall be an Independent layer info SEI message whose nuh\_layer\_id is equal to the second video object identifier.
* The Independent layer info SEI message 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 message whose nuh\_layer\_id is equal to the second video object identifier.

##### Constraints for the stacking function

A VVC input media 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 media 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 media stream is equal to the number of VCL NAL units in the input media stream.
* For each VCL NAL unit in the output media stream, there shall exist a VCL NAL unit in the input media stream that is bit exact identical.
* There shall be an Independent layer info SEI message whose nuh\_layer\_id is equal to the first video object identifier.
* There shall be an Independent layer info SEI message whose nuh\_layer\_id is equal to the second video object identifier.
* The Independent layer info SEI message 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 message whose nuh\_layer\_id is equal to the second video object identifier.

## Instantiation for ISO/IEC 23094-1 Essential Video Coding (EVC)

### General

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

Table 8 provides the bindings of a concept of this document with a concept specified in the EVC specification.

Table 8 — Correspondence between VDI 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) |

### Media and elementary streams constraints

#### General media stream constraints

An EVC media stream used as an instantiation of the media stream in 7.1 Operations on input media 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

#### Media and elementary stream constraints for input formatting functions

##### Constraints for the filtering function

An EVC input media 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 media 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 media 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.

##### Constraints for the inserting function

Two EVC input media 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 media 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 media stream has the same identifier than a SPS or PPS in the second input media stream, then those two SPSs or two PPSs shall have the same payload.

An EVC media stream generated as output of the inserting function shall comply to the following rules:

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

##### Constraints for the appending function

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

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

##### Constraints for the stacking function

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

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

1. (normative)  
     
   Control interface IDL definition
   1. General

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

The control interface is available as electronic attachment at <http://mpegx.int-evry.fr/software/MPEG/Systems/VideoDecodingInterface/specification>.

[Editor’s note] An ISO-hosted link will be provided for publication.

In case of disparity, the control interface available as electronic attachment takes precedence over the reproduction provided in A.2.

* 1. Interface definition

The IDL document is reproduced in Table A.1.

1. Control interface IDL document

|  |
| --- |
| interface MPEG\_VDI {  const unsigned long CAP\_INSTANCES\_FLAG = 0x1;  const unsigned long CAP\_BUFFER\_MEMORY\_FLAG = 0x2;  const unsigned long CAP\_BITRATE\_FLAG = 0x4;  const unsigned long CAP\_MAX\_SAMPLES\_SECOND\_FLAG = 0x8;  const unsigned long CAP\_MAX\_PERFORMANCE\_POINT\_FLAG = 0xA;  enum ConfigParameters {  CONFIG\_OUTPUT\_BUFFER  };  enum ExtParameters {  PARAM\_SUBFRAME\_OUTPUT,  PARAM\_METADATA\_CALLBACK,  PARAM\_OUTPUT\_CROP,  PARAM\_MAX\_OFFTIME\_JITTER  };  enum SampleFormat {  SF\_SCALAR,  SF\_VEC2,  SF\_VEC3,  SF\_VEC4  };  enum SampleType {  ST\_BYTE,  ST\_UNSIGNED\_BYTE,  ST\_SHORT,  ST\_UNSIGNED\_SHORT,  ST\_UNSIGNED\_INT,  ST\_FLOAT  };  struct ConfigDataParameters {  SampleFormat sample\_format;  SampleType sample\_type;  unsigned long sample\_stride;  unsigned long line\_stride;  unsigned long buffer\_offset;  };  struct PerformancePoint {  float picture\_rate;  unsigned long width;  unsigned long height;  unsigned long bit\_depth;  };  struct AggregateCapabilities {  unsigned long flags;  unsigned long max\_instances;  unsigned long buffer\_memory;  unsigned long bitrate;  unsigned long max\_samples\_second;  PerformancePoint max\_performance\_point;  };  exception ErrorAllocation {  string reason;  };  exception ErrorConfig {  string reason;  };  exception ErrorParameter {  string reason;  };  AggregateCapabilities queryCurrentAggregateCapabilities (  in string component\_name,  in unsigned long flags  );  unsigned long getInstance(  in string component\_name,  in unsigned long group\_id // optional, default value = -1  ) raises(ErrorAllocation);  boolean setConfig (  in unsigned long instance\_id,  in ConfigParameters config\_parameters,  in ConfigDataParameters config\_data\_parameters  ) raises(ErrorConfig);  any getParameter (  in unsigned long instance\_id,  in ExtParameters ext\_parameters,  out any parameter  );  boolean setParameter (  in unsigned long instance\_id,  in ExtParameters ext\_parameters,  in any parameter  ) raises(ErrorParameter);  }; |

1. (normative)  
     
   OpenMAX IL VDI extension header
   1. General

The control interface to the video decoding engine is defined for the Open MAX IL interface.

The source code of the extension is available as electronic attachment at <http://mpegx.int-evry.fr/software/MPEG/Systems/VideoDecodingInterface/specification>.

[Editor’s note] An ISO-hosted link will be provided for publication.

In case of disparity, the source code available as electronic attachment takes precedence over the reproduction provided in B.2.

* 1. Source code

The source code is reproduced in Table B.1.

1. OpenMAX IL VDI extension header source code

|  |
| --- |
| /\*  \* The Clear BSD License  \*  \* Copyright (c) 2021 ISO/IEC  \* All rights reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted (subject to the limitations in the disclaimer  \* below) provided that the following conditions are met:  \*  \* \* Redistributions of source code must retain the above copyright notice,  \* this list of conditions and the following disclaimer.  \*  \* \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the distribution.  \*   \* \* Neither the name of the copyright holder nor the names of its  \* contributors may be used to endorse or promote products derived from this  \* software without specific prior written permission.  \*  \* NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY  \* THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND  \* CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A  \* PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR  \* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  \* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  \* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR  \* BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER  \* IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  \* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  \* POSSIBILITY OF SUCH DAMAGE.  \*/  /\*  \* OMX\_MPEG.h - Extension for OpenMax IL version 1.2.0 defined in ISO/IEC 23090-13.  \* The structures and methods are needed for executing the functionalities of  \* multi decoder instances management as defined in ISO/IEC 23090-13.  \*/  #ifndef OMX\_MPEG\_h #define OMX\_MPEG\_h  #ifdef \_\_cplusplus extern "C" { #endif /\* \_\_cplusplus \*/   /\* Each OMX header must include all required header files to allow the  \* header to compile without errors. The includes below are required  \* for this header file to compile successfully   \*/ #include <OMX\_Types.h> #include <OMX\_Core.h>  typedef enum OMX\_MPEG\_ERRORTYPE {  OMX\_ErrorMpegNone = 0,  OMX\_ErrorVdiNotImplemented = OMX\_ErrorVendorStartUnused + 1,  OMX\_ErrorVdiAllocation,  OMX\_ErrorVdiParameter,  OMX\_ErrorVdiConfig,  OMX\_ErrorMpegMax = 0X7FFFFFFF } OMX\_MPEG\_ERRORTYPE;  const OMX\_U32 CAP\_INSTANCES\_FLAG = 1 << 0; const OMX\_U32 CAP\_BUFFER\_MEMORY\_FLAG = 1 << 1; const OMX\_U32 CAP\_BITRATE\_FLAG = 1 << 2; const OMX\_U32 CAP\_MAX\_SAMPLES\_SECOND\_FLAG = 1 << 3; const OMX\_U32 CAP\_MAX\_PERFORMANCE\_POINT\_FLAG = 1 << 4;  typedef enum OMX\_MPEG\_SAMPLEFORMAT {  OMX\_SampleSFormatScalar = 1,  OMX\_SampleFormatVec2,  OMX\_SampleFormatVec3,  OMX\_SampleFormatVec4,  OMX\_SampleFormatMax = 0X7FFFFFFF } OMX\_MPEG\_SAMPLEFORMAT;  typedef enum OMX\_MPEG\_SAMPLETYPE {  OMX\_SampleTypeByte = 5120,  OMX\_SampleTypeUnsigned\_Byte,  OMX\_SampleTypeShort,  OMX\_SampleTypeUnsignedShort,  OMX\_SampleTypeUnsigned\_Int = 5125,  OMX\_SampleTypeFloat,  OMX\_SampleTypeMax = 0X7FFFFFFF } OMX\_MPEG\_SAMPLETYPE;  typedef enum OMX\_CONFIG\_MPEG\_PARAMETERS {  OMX\_ConfigOutputBuffer = OMX\_IndexVendorStartUnused + 1001 } OMX\_CONFIG\_MPEG\_PARAMETERS;  typedef enum OMX\_MPEG\_EXTPARAMETERS {  OMX\_ExtParamSubframeOutput = 1001,  OMX\_ExtParamMetadataCallback,  OMX\_ExtParamOutputCrop,  OMX\_ExtParamMaxOfftimeJitter,  OMX\_ExtParamMax = 0X7FFFFFFF } OMX\_MPEG\_EXTPARAMETERS;  typedef struct OMX\_CONFIG\_MPEG\_DATAPARAMETERS {  OMX\_MPEG\_SAMPLEFORMAT eSampleFormat;  OMX\_MPEG\_SAMPLETYPE eSampleType;  OMX\_U32 nSampleStride;  OMX\_U32 nLineStride;  OMX\_U32 nBufferOffset; } OMX\_CONFIG\_MPEG\_DATAPARAMETERS;  typedef struct OMX\_MPEG\_PERFORMANCEPOINT {  float fPictureRate;  OMX\_U32 nWidth;  OMX\_U32 nHeight;  OMX\_U32 nBitDepth; } OMX\_MPEG\_PERFORMANCEPOINT;  typedef struct OMAX\_MPEG\_AGGREGATECAPABILITIES {  OMX\_U32 nFlags;  OMX\_U32 nMaxInstances;  OMX\_U32 nBufferMemory;  OMX\_U32 nBitrate;  OMX\_U32 nMaxSamplesSecond;  OMX\_MPEG\_PERFORMANCEPOINT eMaxPerformancePoint; } OMAX\_MPEG\_AGGREGATECAPABILITIES;  OMX\_API OMAX\_MPEG\_AGGREGATECAPABILITIES\* OMX\_APIENTRY OMX\_MPEG\_QueryCurrentAggregateCapabilities(  OMX\_IN OMX\_STRING cComponentName,  OMX\_IN OMX\_U32 nFlags);  OMX\_API OMX\_MPEG\_ERRORTYPE OMX\_APIENTRY OMX\_MPEG\_GetInstance(  OMX\_IN OMX\_STRING cComponentName,  OMX\_IN OMX\_U32 nGroupId);  OMX\_API OMX\_MPEG\_ERRORTYPE OMX\_APIENTRY OMX\_MPEG\_SetConfig(  OMX\_IN OMX\_U32 nInstanceId,  OMX\_IN OMX\_CONFIG\_MPEG\_PARAMETERS eConfigParameters,  OMX\_IN OMX\_CONFIG\_MPEG\_DATAPARAMETERS eConfigDataParameters);  OMX\_API OMX\_MPEG\_ERRORTYPE OMX\_APIENTRY OMX\_MPEG\_GetParameter(  OMX\_IN OMX\_U32 nInstanceId,  OMX\_IN OMX\_MPEG\_EXTPARAMETERS eExtParameters,  OMX\_OUT void\* pParameter );  OMX\_API OMX\_MPEG\_ERRORTYPE OMX\_APIENTRY setParameter(  OMX\_IN OMX\_U32 nInstanceId,  OMX\_IN OMX\_MPEG\_EXTPARAMETERS eExtParameters,  OMX\_IN void\* pParameter);  #ifdef \_\_cplusplus } #endif /\* \_\_cplusplus \*/  #endif /\* File EOF \*/ |

1. (normative)  
     
   Supplemental Enhancement Information (SEI) syntax and semantics
   1. **VDI SEI envelope**
      1. **General**

Clause C.1 defines a generic envelope for carrying SEI messages defined in this specification. Some of the VDI SEI messages may apply to only certain video coding specification.

The VDI SEI envelope is registered as SEI payload in the following video coding specifications:

* ISO/IEC 23090-3 VVC

[Editor’s note] Based on the conclusions (see below) of the joint meeting between WG3 and WG5 at MPEG #135, a VVC amendment will be started at the next meeting to register VDI SEI envelope. Here are the notes from the meeting:

* *Codepoints for Green Metadata and VDI: Better define separate ones in VVC, and refer the corresponding systems specs directly*
* *WG3 would have the primary responsibility what is going in*
* *JVET should start a VVC amendment in October or January to align with the progression in WG3 (expecting this could be final at DAM stage)*
  + 1. **VDI SEI envelope syntax**

1. Syntax of VDI SEI envelope

|  |  |
| --- | --- |
| **Syntax** | **Descriptor** |
| vdi\_sei\_envelope( payloadSize ) { |  |
| **vdi\_sub\_type** | u(8) |
| if(vdi\_sub\_type == 0) |  |
| independent\_layer\_info(payloadSize - 1) |  |
| else |  |
| reserved\_message(payloadSize - 1) |  |
| } |  |

* + 1. **VDI SEI envelope semantics**

**vdi\_sub\_type** indicates the payload type carried in the VDI SEI envelope.

* 1. **Independent layer info SEI** **message**
     1. **Independent layer info SEI message syntax**

1. Syntax of independent layer info SEI message

|  |  |
| --- | --- |
| **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.



1. 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 the 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 message, parse the payload and store the boundary identifiers for the corresponding layer.
  + If any of VPS, PPS or Independent layer info SEI message 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() function. Only one buffer per tunnel is allowed and one of the two components acts a supplier of that buffer.

**Error! Reference source not found.** depicts an example of connected components and the buffer usage:

A screenshot of a map

Description automatically generated

1. 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. **Error! Reference source not found.** shows an example:

A screenshot of a cell phone

Description automatically generated

1. 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.

1. Format of the OpenMAX IL buffer header

|  |
| --- |
| typedef struct OMX\_BUFFERHEADERTYPE  {  OMX\_U32 nSize;  OMX\_VERSIONTYPE nVersion;  OMX\_U8\* pBuffer;  OMX\_U32 nAllocLen;  OMX\_U32 nFilledLen;  OMX\_U32 nOffset;  OMX\_PTR pAppPrivate;  OMX\_PTR pPlatformPrivate;  OMX\_PTR pInputPortPrivate;  OMX\_PTR pOutputPortPrivate;  OMX\_HANDLETYPE hMarkTargetComponent;  OMX\_PTR pMarkData;  OMX\_U32 nTickCount;  OMX\_TICKS nTimeStamp;  OMX\_U32 nFlags;  OMX\_U32 nOutputPortIndex;  OMX\_U32 nInputPortIndex;  } OMX\_BUFFERHEADERTYPE; |

1. Buffer flags defined in OpenMAX IL

|  |
| --- |
| #define OMX\_BUFFERFLAG\_EOS 0x00000001  #define OMX\_BUFFERFLAG\_STARTTIME 0x00000002  #define OMX\_BUFFERFLAG\_DECODEONLY 0x00000004  #define OMX\_BUFFERFLAG\_DATACORRUPT 0x00000008  #define OMX\_BUFFERFLAG\_ENDOFFRAME 0x00000010  #define OMX\_BUFFERFLAG\_SYNCFRAME 0x00000020  #define OMX\_BUFFERFLAG\_EXTRADATA 0x00000040  #define OMX\_BUFFERFLAG\_CODECCONFIG 0x00000080  #define OMX\_BUFFERFLAG\_TIMESTAMPINVALID 0x00000100  #define OMX\_BUFFERFLAG\_READONLY 0x00000200  #define OMX\_BUFFERFLAG\_ENDOFSUBFRAME 0x00000400  #define OMX\_BUFFERFLAG\_SKIPFRAME 0x00000800 |

* + 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.