International Standard

ISO/IEC 23090-13

First edition
2024-01

Information technology — Coded representation of immersive media —

Part 13: Video decoding interface for immersive media

Technologies de l'information — Représentation codée de média immersifs —

Partie 13: Interface de décodage vidéo pour les média immersifs

Reference number
ISO/IEC 23090-13:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

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 document 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 or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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. In the IEC, see www.iec.ch/understanding-standards.

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

A list of all parts in the ISO 23090 series can be found on the ISO and IEC websites.

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 and www.iec.ch/national-committees.

Introduction

The interfaces and operations specified in this document come as extensions of existing video decoding engine specifications exposing hardware video decoding capabilities.

Information technology — Coded representation of immersive media —

Part 13: Video decoding interface for immersive media

1 Scope

This document specifies the interfaces of a video decoding engine as well as the operations related to elementary streams and metadata that can be performed by this video decoding engine. To support those operations, this document also specifies SEI messages when necessary for certain video codecs.

2 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 23094-1, *Information technology — General video coding — Part 1: Essential video coding*

3 Terms and definitions

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

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
media stream
part of an *elementary stream* (3.2) or 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.

Note 2 to entry: A media stream may contain metadata such as non-VCL NAL units.

3.2
subframe
independently decodable unit smaller than a frame to which post-decoding processing by the decoder, if any, has been applied

3.3
video object
independently decodable substream of a video *elementary stream* (3.2)

3.4
video object identifier
integer identifying a *video object* (3.4)

4 Abbreviated terms

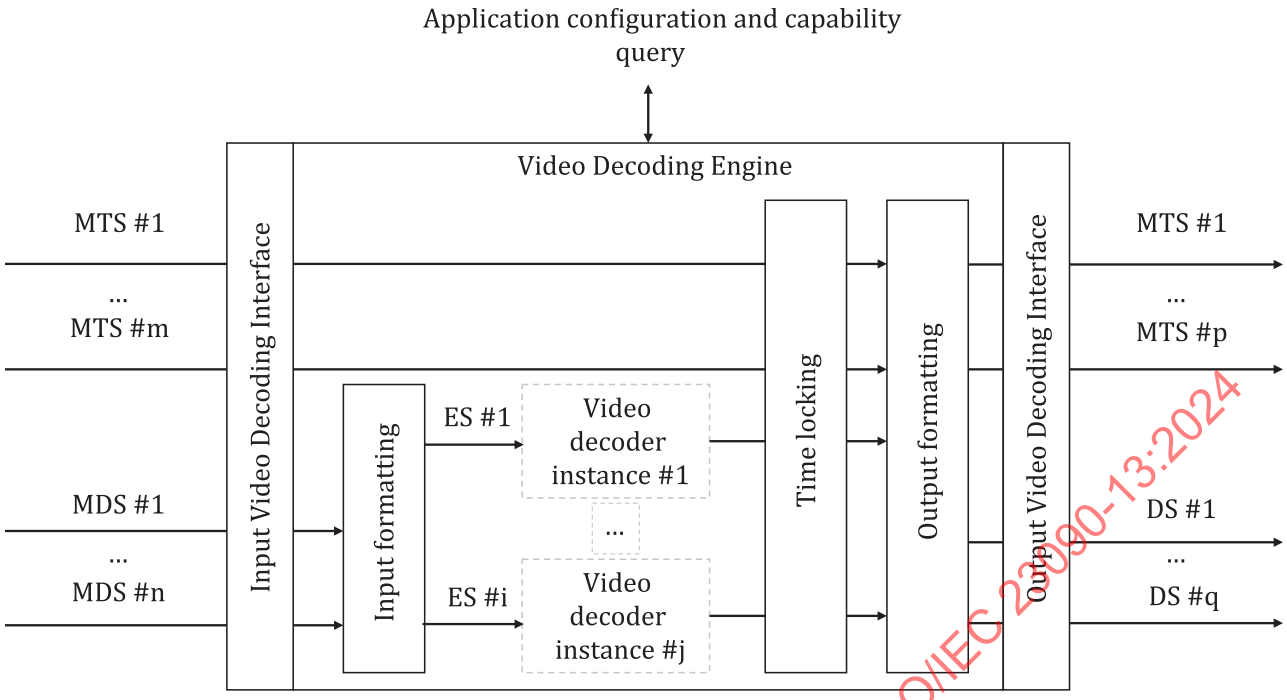
API	application programming interface
ES	elementary stream
I	video object identifier
IDL	interface definition language
IVDI	input video decoding interface
MDS	media stream
NAL	network abstraction layer
OLS	output layer set
OVDI	output video decoding interface
PPS	picture parameter set
SEI	supplemental enhancement information
SPS	sequence parameter set
VCL	video coding layer
VDE	video decoding engine

5 Video decoding engine

5.1 General

The video decoding engine (VDE) enables the decoding, the synchronization and the formatting of media streams which are one or more aggregated elementary streams or a part thereof. The media streams are 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. Between the input and the output, the VDE extracts and merges independently decodable regions from a set of input media streams via the input formatting function and generates a set of elementary streams fed to the video decoder instances which run inside the engine. The VDE can execute a merging operation or an extraction operation on the input media streams such that the number of running video decoder instances is different from the number of input media streams required by the application. For example, a VDE can be incapable of decoding a single 4K input media stream with one decoder instance, but it can decode some of the independently decodable regions, at a lower resolution, present in that input media stream. To this end, the VDE should first verify the availability of sufficient resources to run in parallel those video decoder instances.

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
m	number of input metadata streams
n	number of media streams
j	number of video decoder instances
p	number of output metadata streams
q	number of decoded sequences

Figure 1 — Video decoding engine and interfaces

NOTE 1 Multiple elementary streams that are output of the input formatting function can be fed to a single video decoder instance.

NOTE 2 The concept of metadata stream does not yet possess a definition in this document. 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 as several decoder instances each with their own interface.

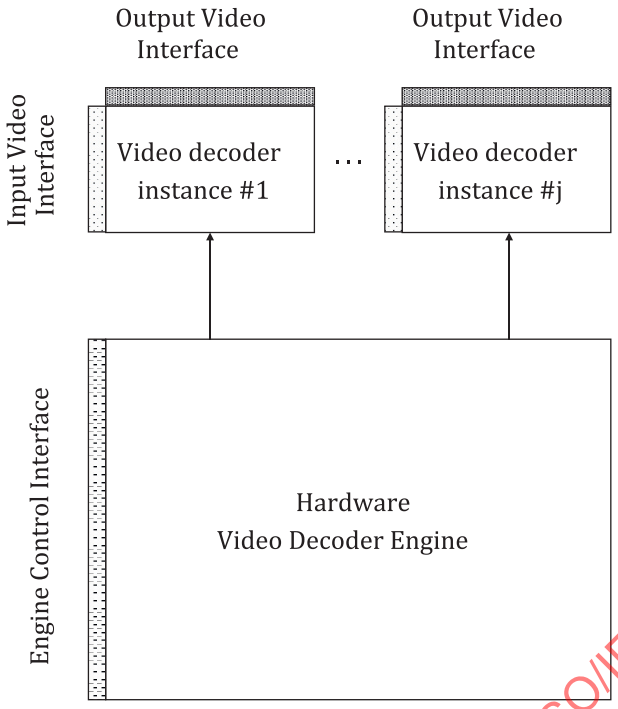


Figure 2 — Example relationship between video decoder instances and video decoder hardware engine

5.2 Input video decoding interface

The video decoding engine accepts media streams and metadata streams. There is at least one media stream as input but there is no constraint on the number of metadata streams with respect to the number of media streams being concurrently consumed by the VDE.

The input of the VDE comprises thus:

— `n` media streams

— `m` metadata streams

5.3 Output video decoding interface

The video decoding engine outputs decoded video sequences and 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 a 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

5.4 Control interface to the Video Decoding Interface

5.4.1 Functions

In order to support immersive media applications, subclause 5.4 defines an abstract video decoding interface. A video decoding platform that complies with this document shall implement this video decoding interface whose IDL can be found in Annex A.

The video decoding interface consists of the abstract functions defined in the following subclause. These functions are defined using the IDL syntax specified in ISO/IEC 19516.

Figure 3 depicts an example instantiation of decoder instances using some of the functionalities of the video decoding interface. The video decoder instances with identifiers 1 to 3 belong to the group with the identifier 4. By this grouping mechanism, the three instances write the decoded sequences into a single aggregate buffer and the decoding operations across those instances are performed in a coordinated manner such that no instance runs ahead or behind the others.

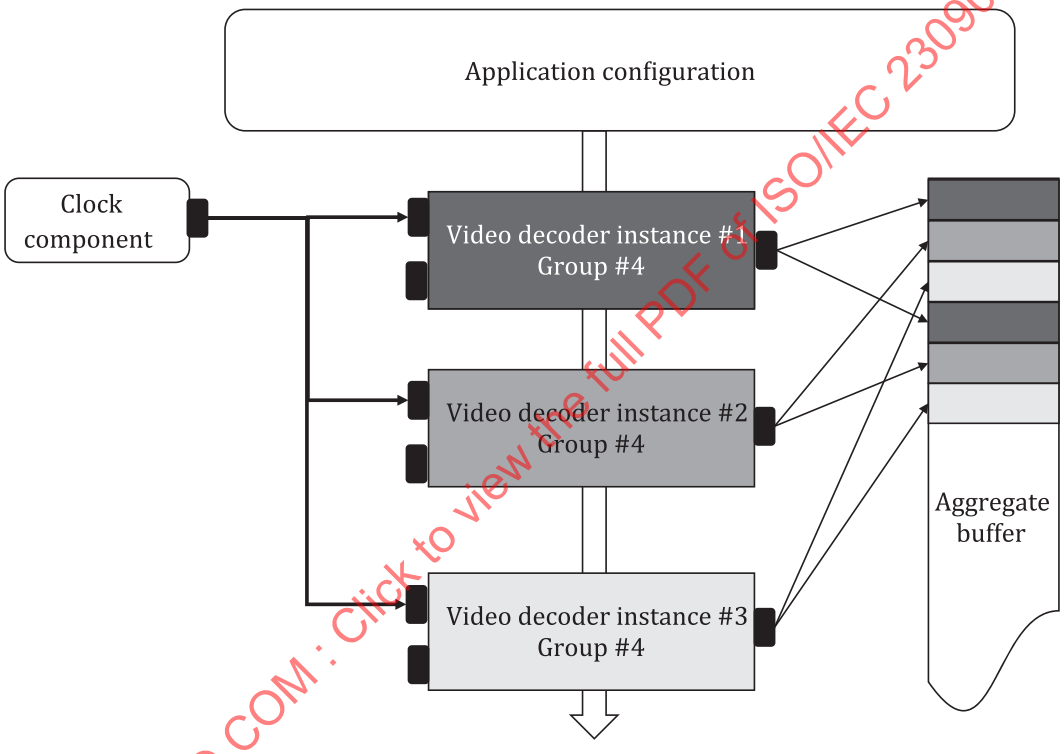


Figure 3 — Example instantiation using VDI

5.4.1.1 queryCurrentAggregateCapabilities()

5.4.1.1.1 Declaration

The IDL declarations of the `queryCurrentAggregateCapabilities()` function along with the `AggregateCapabilities` and `PerformancePoint` structures and the capabilities flags are defined as follows:

```
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
);
```

5.4.1.1.2 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 capability flags below can set separately or in a single function call to query one or more parameters.

The `component_name` provides the name of the 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 is rather for all the components of the decoding platform. Components are hardware or software functionalities exposed by the Video Decoding Engine such as decoders.

`CAP_INSTANCES_FLAG` queries the `max_instances` parameter which indicates the maximum number of decoder instances that can be instantiated at this moment for the provided decoder component.

`CAP_BUFFER_MEMORY_FLAG` queries the `buffer_memory` parameter which indicates the instantaneous global maximum available buffer size in bytes that can be allocated independently of any components at this moment on the decoder platform for buffer exchange. The allocation of the memory can be done by the application or the VDE itself depending on the VDE instantiation.

`CAP_BITRATE_FLAG` queries the `bitrate` parameter which indicates the instantaneous maximum coded bitrate in bits per second that the queried component can process.

`CAP_MAX_SAMPLES_SECOND_FLAG` queries the `max_samples_second` parameter which indicates the instantaneous maximum number of luma and chroma samples combined per second that the queried component is able to process.

`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 `PerformancePoint` contains the following parameters:

— `picture_rate` indicating the instantaneous 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 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.

5.4.1.2 getInstance()

5.4.1.2.1 Declaration

The IDL declarations of the `getInstance()` function and the associated `ErrorAllocation` exception are defined as follows:

```
exception ErrorAllocation {
    string reason;
};

unsigned long getInstance(
    in string component_name,
    in unsigned long group_id // optional, default value = -1
) raises(ErrorAllocation);
```

5.4.1.2.2 Definition

The result of a successful call to the `getInstance()` function call shall provide the identifier of the instance and the `group_id` that is assigned or created for this new instance, if one was requested. The default behavior is that the decoder instance does not belong to any already established group but is assigned to a newly created group.

Several decoder instances belonging to a same group means that the VDE treats those instances collectively such that the decoding states of those instances progress in synchrony and not in competition against each other. As a result, the VDE will also ensure synchronized output writing operation, possibly into an aggregate buffer. There are no conditions for two video decoder instances to be in the same group.

5.4.1.3 setConfig()

5.4.1.3.1 Declaration

The IDL declarations of the `setConfig()` function, the associated `ErrorConfig` exception, the `ConfigDataParameters` structure and the `ConfigParameters` enumeration are defined as follows:

```
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);
```

5.4.1.3.2 Definition

The `setConfig()` function may be called with the 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.

5.4.1.4 getParemeter() and setParameter()

5.4.1.4.1 Declaration

The IDL declarations of the `getParameter()` and `setParemeter()` functions as well as the associated `ErrorParameter` exception and the `ExtParameters` enumeration are defined as follows:

```
    enum ExtParameters {
        PARAM_PARTIAL_OUTPUT,
        PARAM_SUBFRAME_OUTPUT,
        PARAM_METADATA_CALLBACK,
        PARAM_OUTPUT_CROP,
        PARAM_OUTPUT_CROP_WINDOW,
        PARAM_MAX_OFFTIME_JITTER
};

    struct CropWindow {
        unsigned long x;
        unsigned long y;
        unsigned long width;
        unsigned long height;
    };

    exception ErrorParameter {
        string reason;
    };

    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);
```

5.4.1.4.2 Definition

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

`PARAM_PARTIAL_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_SUBFRAME_OUTPUT` indicates the one or more subframes to be output by the decoder.

`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_OUTPUT_CROP_WINDOW` indicates the part of the decoded frame to be cropped and output.

`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 instances, which requires context switching between the different decoder instances.

5.5 Examples of video decoding engine instantiations

5.5.1 Mapping on OpenMAX™ integration layer (OpenMAX IL)

5.5.1.1 Overview

For more information on OpenMAX[1)] IL, Annex E provides a brief description of the main functions of this API.

5.5.1.2 Mapping of VDI functions

The function defined in 5.4 are mapped on the OpenMAX IL interface by using the extension mechanism defined by the specification. This MPEG VDI extension for OpenMAX IL is formatted as a C header file and registered with the vendor name “MPEG”.

Annex B defines the MPEG VDI extension for OpenMAX IL and provides information to access the electronic version of this extension.

5.5.2 Mapping on Vulkan® Video

5.5.2.1 Overview

Vulkan®[2)] Video (VK) is an extension of the Vulkan API which defines functions exposed by Graphics Processing Units (GPU). This extension provides interfaces for an application to leverage hardware decoding and encoding capabilities present on GPUs.

A VK Video Session consists of a single decoding session on a single layer. As a result, a single VK Video Session corresponds to a single video decoder instance depicted in Figure 1.

The mapping of VDI functions on VK is summarised in Table 1.

1) OpenMAX™ is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO IEC of this product.

2) Vulkan® is an example of a suitable product available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO IEC of this product.

Table 1 — Summary of VDI function mapping on Vulkan® Video

VDI functions	VK mapping
queryCurrent AggregateCapabilities	New `vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG()` function
getInstance (grouping)	Extending `VkVideoSessionCreateInfoKHR` with a group identifier passed in the new structure `VkVideoSessionCreateInfoGroupingMPEG`. Call of existing `vkCreateVideoSessionKHR()`.
setConfig (buffer configuration)	Mapping on existing `VkVideoSessionCreateInfoKHR` and `VkVideoPictureResourceKHR` structures.
getParameter and **setParameter**	New `VkVideoSessionOutputParameterMPEG` structure

5.5.2.2 The vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG() function

5.5.2.2.1 Definition

The VK Video API provides a function for querying capabilities for a single VK Video Profile which is called `vkGetPhysicalDeviceVideoCapabilitiesKHR()`. Similar to this function, the VDI VK mapping defines the `vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG()` function. In contrast to the `vkGetPhysicalDeviceVideoCapabilitiesKHR()` function, the `vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG()` function allows to query the aggregates capabilities of the physical device. When it is called with a certain profile, the aggregated capabilities pertains to this given profile.

5.5.2.2.2 Declaration

```
VkResult vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG(
    VkPhysicalDevice                physicalDevice,
    VkVideoProfileKHR*              pVideoProfile,
    VkCurrentVideoCapabilitiesMPEG*    pCapabilities);
```

5.5.2.2.3 Semantics

`physicalDevice` is the physical device whose video decode or encode capabilities are to be queried.

`pVideoProfile` is a pointer to a `VkVideoProfileKHR` structure with a chained codec-operation specific video profile structure.

`pCapabilities` is a pointer to a `VkCurrentVideoCapabilitiesMPEG` structure in which the capabilities are returned.

5.5.2.3 The VkCurrentVideoCapabilitiesMPEG structure

5.5.2.3.1 Definition

The `VkCurrentVideoCapabilitiesMPEG` structure holds the information returned by a call to the `vkGetPhysicalDeviceCurrentVideoCapabilitiesMPEG()` function defined in subclause 5.5.2.2.

5.5.2.3.2 Declaration

```
typedef struct VkCurrentVideoCapabilitiesMPEG {
    VkStructureType         sType;
    void*                   pNext;
    uint32_t                maxInstances;
    uint32_t                bufferMemory;
    uint32_t                bitrate;
    uint32_t                maxSamplesSecond;
    VkPerformancePointMPEG*      maxPerformancePoint;
```

```
} VkCurrentVideoCapabilitiesMPEG;
```

5.5.2.3.3 Semantics

`sType` is the type of this structure.

`pNext` is NULL or a pointer to a structure extending this structure.

`maxInstances` see semantic in subclause 5.4.1.1.2.

`bufferMemory` see semantic in subclause 5.4.1.1.2.

`bitrate` see semantic in subclause 5.4.1.1.2.

`maxSamplesSecond` see semantic in subclause 5.4.1.1.2.

`maxPerformancePoint` is a pointer to a `VkPerformancePointMPEG` structure in which the properties of the maximum performance are returned.

5.5.2.4 The VkCurrentVideoCapabilitiesMPEG structure

5.5.2.4.1 Definition

The `VkCurrentVideoCapabilitiesMPEG` structure contains properties describing a performance point for a video processing entity.

5.5.2.4.2 Declaration

```
typedef struct VkPerformancePointMPEG {
    VkStructureType sType;
    void*           pNext;
    uint32_t        pictureRate;
    uint32_t        height;
    uint32_t        width;
    uint32_t        bitDepth;
} VkPerformancePointMPEG;
```

5.5.2.4.3 Semantics

`sType` is the type of this structure.

`pNext` is NULL or a pointer to a structure extending this structure.

`pictureRate` see semantic in subclause 5.4.1.1.2.

`height` see semantic in subclause 5.4.1.1.2.

`width` see semantic in subclause 5.4.1.1.2.

`bitDepth` see semantic in subclause 5.4.1.1.2.

5.5.2.5 The VkVideoSessionCreateInfoGroupingMPEG structure

5.5.2.5.1 Definition

The `VkVideoSessionCreateInfoGroupingMPEG` structure allows to attach a group identifier to a video decoding instance created via the VK Video API. This structure extends the `VkVideoSessionCreateInfoKHR` structure defined in the VK Video API

5.5.2.5.2 Declaration

```
typedef struct VkVideoSessionCreateInfoGroupingMPEG {
    VkStructureType                 sType;
    const void*                     pNext;
    uint32_t                        groupId;
} VkVideoSessionCreateInfoGroupingMPEG;
```

5.5.2.5.3 Semantics

`sType` is the type of this structure.

`pNext` is NULL or a pointer to a structure extending this structure.

`groupId` see semantic in subclause 5.4.1.2.2.

5.5.2.6 The VkVideoSessionOutputParameterMPEG structure

5.5.2.6.1 Definition

The `VkVideoSessionOutputParameterMPEG` structure contains parameters that configure the properties of the output of the VK Video Session.

5.5.2.6.2 Declaration

```
typedef struct VkVideoSessionOutputParameterMPEG {
    VkStructureType sType;
    const void*     pNext;
    VkFlag          partialOutput;
    uint32_t*       subframeCount;
    uint32_t*       pSubframeOutput;
    VkFlag          outputCrop;
    VkExtent2D*     pOutputCropWindow;
    uint32_t        maxOfftimeJitter;
    void*           pMetadataCallback;
} VkVideoSessionOutputParameterMPEG;
```

5.5.2.6.3 Semantics

`sType` is the type of this structure.

`pNext` is NULL or a pointer to a structure extending this structure.

`partialOutput` see semantic in subclause 5.4.1.4.2.

`subframeCount` and `pSubframeOutput` see semantic in subclause 5.4.1.4.2.

`outputCrop` see semantic in subclause 5.4.1.4.2.

`pOutputCropWindow` see semantic in subclause 5.4.1.4.2.

`maxOfftimeJitter` see semantic in subclause 5.4.1.4.2.

`pMetadataCallback` see semantic in subclause 5.4.1.4.2.

5.5.3 Informative mapping

This specification also provides informative mapping on other APIs such as in Annex F on the media source extension (MSE).

6 VDI systems decoder model

6.1 Introduction

The VDI systems decoder model extends on the systems decoder model (SDM) defined in ISO/IEC 14496-1. Compared to the SDM, the VDI SDM introduces a new interface in addition to the elementary stream interface called the media stream interface. This interface is the input of the Input Formatter, also called input formatting function, which takes as input the so-called media streams. The output of the Input Formatter is one or more elementary streams that can be further passed on to the decoders.

These elements are depicted in Figure 4.

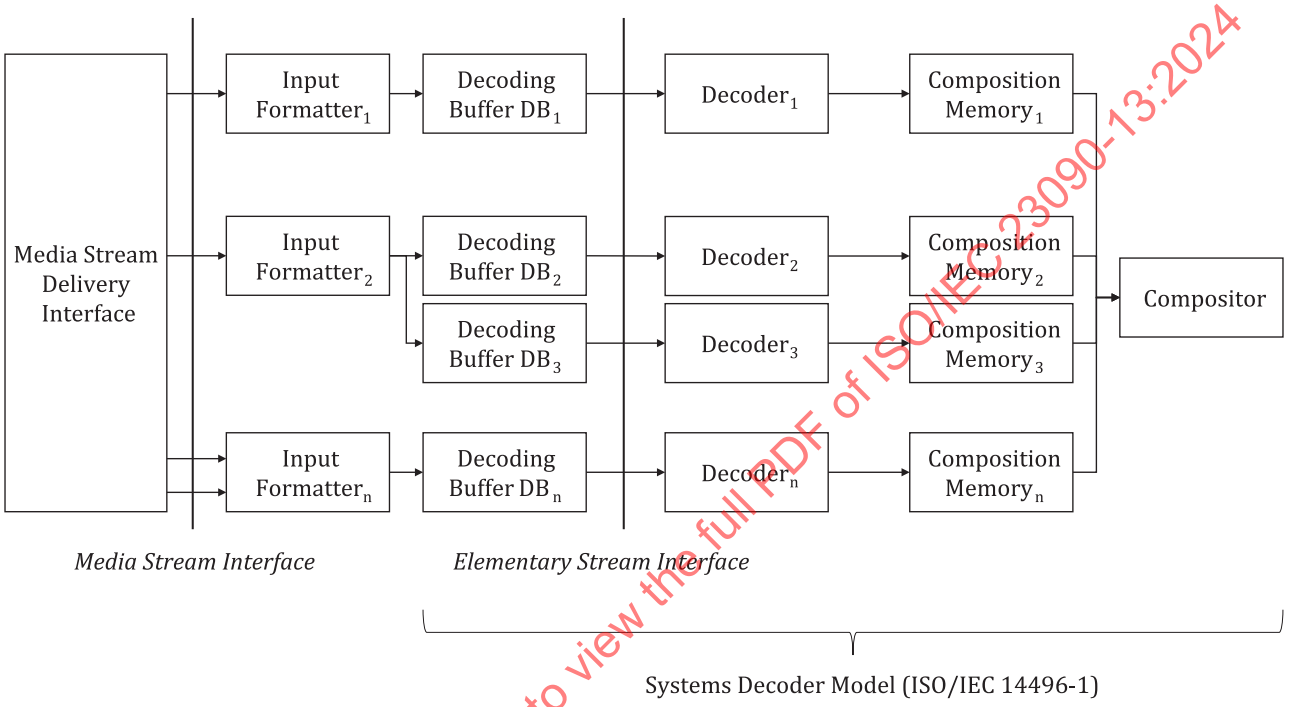


Figure 4 — VDI systems decoder model

6.2 Concepts of the VDI systems decoder model

6.2.1 General

The concepts necessary for the specification are the formatting, the timing, and the buffering model. The sequence of definitions corresponds to a walk from the left to the right side of the VDI SDM illustration in Figure 4.

6.2.2 Media stream

6.2.3 Media stream interface

The media stream interface is a concept that models the exchange of media stream data between the delivery interface and the input formatting function.

6.2.4 Input formatter

The input formatter takes one or more media streams as input and generates one or more elementary streams as output. A single input formatter may be attached to several decoding buffers when it produces individual elementary streams or multi-layer elementary streams.

6.2.5 Access Units (AU)

See 7.1.2.2 in ISO/IEC 14496-1.

6.2.6 Decoding Buffer (DB)

See 7.1.2.4 in ISO/IEC 14496-1.

6.2.7 Elementary Streams (ES)

See 7.1.2.5 in ISO/IEC 14496-1.

6.2.8 Elementary Stream Interface (ESI)

See 7.1.2.6 in ISO/IEC 14496-1.

6.2.9 Decoder

See 7.1.2.7 in ISO/IEC 14496-1.

6.2.10 Composition Units (CU)

See 7.1.2.8 in ISO/IEC 14496-1.

6.2.11 Composition Memory (CM)

See 7.1.2.9 in ISO/IEC 14496-1.

6.2.12 Compositor

See 7.1.2.10 in ISO/IEC 14496-1.

7 Video decoder interface

7.1 General

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 optimization choice for the platform when considering available resources such as computational load, energy consumption, memory, etc. However, the number of input media streams fed through the IVDI depends on the application needs to properly render the media experience. Therefore, one or more input media streams may be fed to the same video decoding instance thanks to the block called "Input formatting" in Figure 1.

This clause defines the binding for several video codecs to realize the operations on input video streams.

7.2 Operations on input media streams

7.2.1 General

The input formatting function in Figure 1 provides several operations on media streams and video objects. The input formatting function results 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 them 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 them as long as the final output consists of valid elementary streams. The actual implementation of those combined operations is out-of-scope of this specification and can be subject to optimization by the implementers. Example of possible implementations are provided in Annex D.

A media stream contains one or more video objects and a video object is contained into one elementary stream. Each video object in an elementary stream provides information for enabling 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.

7.2.2 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

7.2.3 Filtering by video object identifier

7.2.3.1 Definition

Function: Filtering

Definition: $f : MDS \times I \rightarrow ES$

Input: one media stream with at least one video object

the identifier of the selected video object to be extracted

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

Signature:
```
ElementaryStream output_stream_filtering(MediaStream input_stream,
                                         VideoObjectIdentifier id)
```

For each i-th access unit in the input media stream, the function makes a copy of the access unit. Then, the function lists the video object samples present in this copied access unit. If a video object sample does not correspond to the video object identifier passed as input, the video object sample is removed from the copied access unit. Lastly, the copied access unit is appended to the output elementary stream as a new access unit.

NOTE The function 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 and the properties of the input access units are passed on to the access units of the output stream.

7.2.3.2 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 Figure 5 shows the video of the input media stream on the left and the output elementary stream on the right. 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.

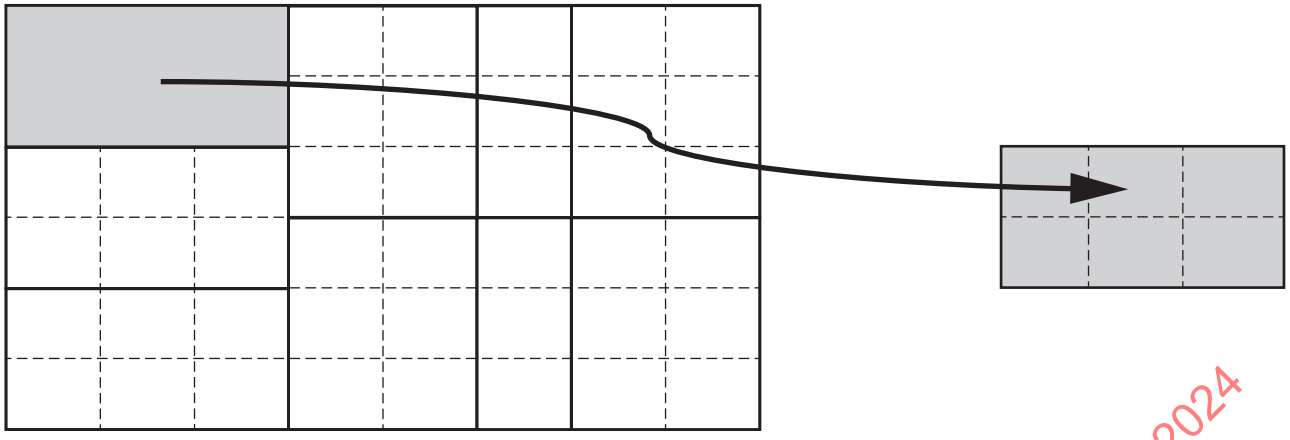


Figure 5 — Example of input and output video for the filtering function

7.2.4 Inserting video objects

7.2.4.1 Definition

Function: Inserting

Definition: $f : MDS \times MDS \rightarrow MDS$

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

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

Signature:
```
MediaStream output_stream inserting(MediaStream input_stream_1,
                                    MediaStream input_stream_2)
```

For each i-th access unit in the first and second input media streams, the function makes a copy of the i-th access unit of the second input media stream. Then, the function lists the video object samples present in the i-th access unit from the first input media stream. Each video object sample is added to the copied access unit. Lastly, the copied access unit is appended to the output media stream as a new access unit.

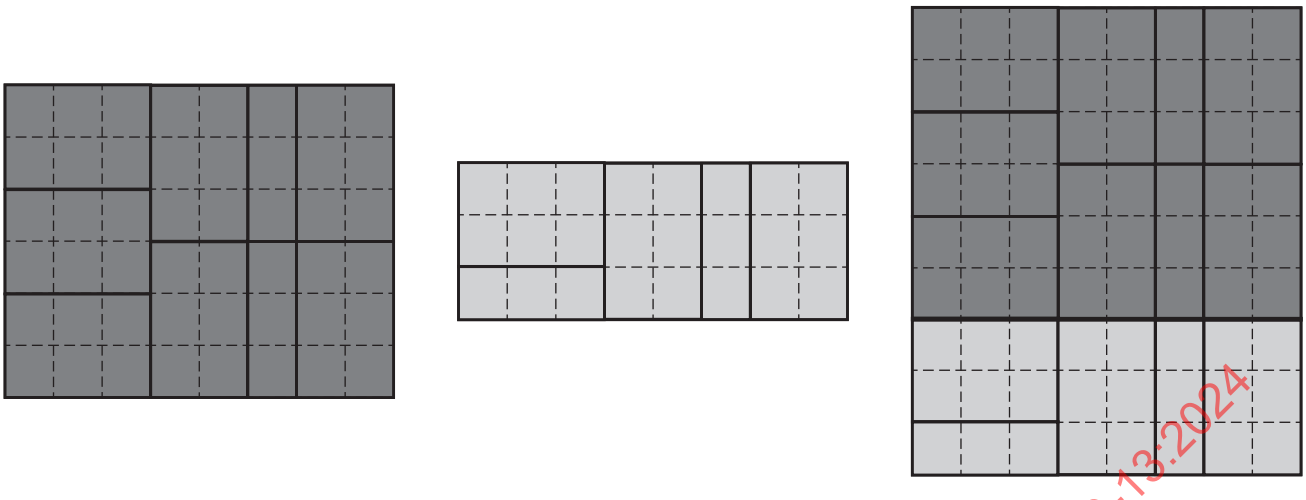
NOTE 1 The inserting operation stops as soon as one of the two input media streams ends.

NOTE 2 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 media stream.

7.2.4.2 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.

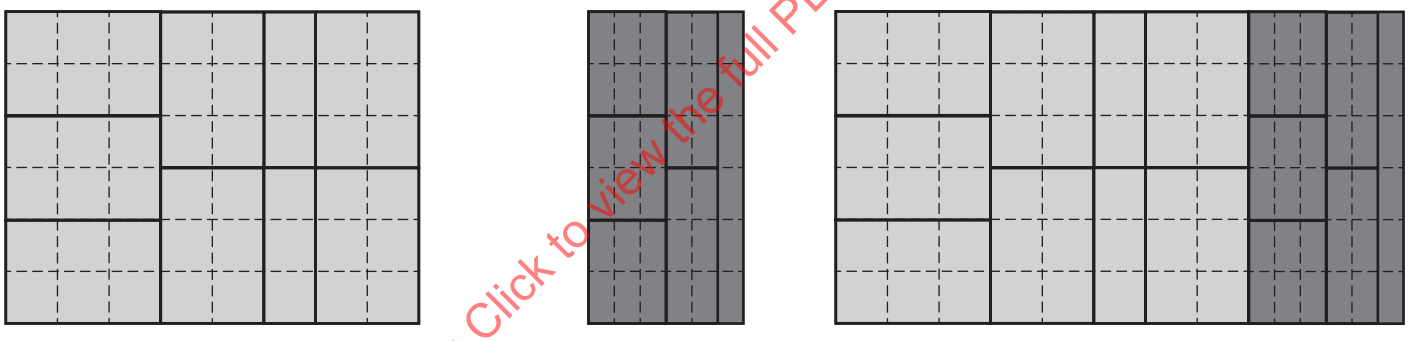
In case the video objects are slices, either the width or the height of the coded pictures of the input media streams are equal in order to maintain the rectangular shape of the video of the output media stream. In case the widths of the two input videos are equal, as shown in the Figure 6 a) and b), then the two videos are vertically stitched as shown in the diagram c). 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.



a) Input media stream one **b) Input media stream two** **c) Output elementary stream**

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

If the heights of two videos are equal, as shown in Figure 7 a) and b), then the two videos are horizontally stitched as shown in the diagram c). 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.



a) Input media stream one **b) Input media stream two** **c) Output elementary stream**

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

7.2.5 Appending two video objects

7.2.5.1 Definition

Function: Appending

Definition: $f : MDS \rightarrow MDS$

Input: One media stream with at least two video objects

Output: One media stream with two video objects which are left and right spatial neighbors

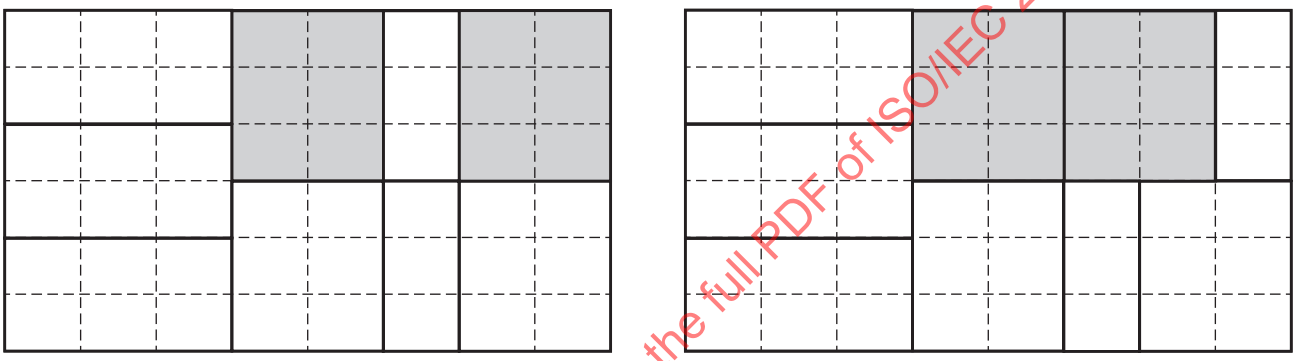
Signature:
```
MediaStream output_stream appending(MediaStream input_stream,
                                    VideoObjectIdentifier object_id_1,
                                    VideoObjectIdentifier object_id_2)
```

For each i-th access unit in the input media stream, the function makes a copy of this i-th access unit. Then, the function sets the position of the video object samples that belong to the video object identified by the second video object identifier right of the video object samples belonging to the object identified by the first video object identifier in this copied access unit. This positioning is done in such a way that the top boundaries of both video object samples are aligned. Lastly, the copied access unit is appended to the output media stream as a new access unit.

7.2.5.2 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 neighbors.

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). During this operation, slice header may need to be updated to correctly signal the changes of the video object identifiers, e.g. the slice addresses.



a) Input media stream **b) Output elementary stream**

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

7.2.6 Stacking two video objects

7.2.6.1 Definition

Function: Stacking

Definition: $f : MDS \rightarrow MDS$

Input: One media stream with at least two video objects

Output: One media stream with two video objects which are top and bottom spatial neighbors

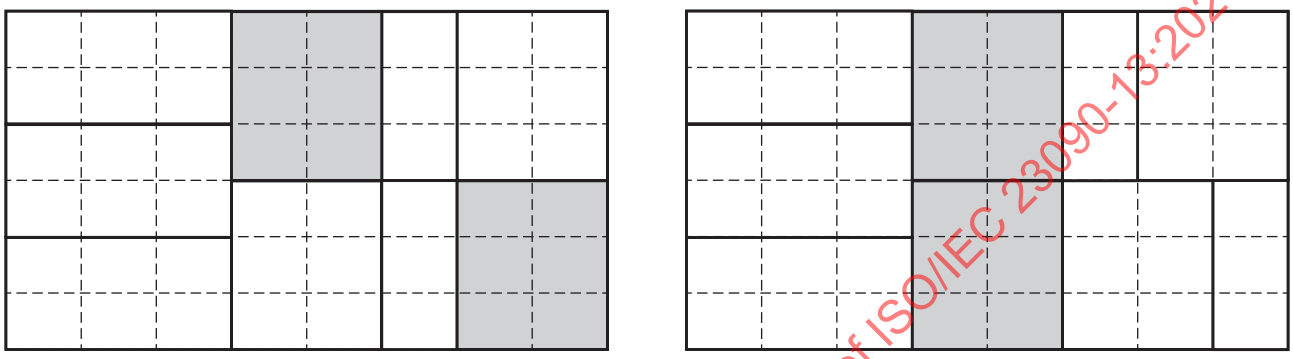
Signature:
```
MediaStream output_stream stacking(MediaStream input_stream,
                                   VideoObjectIdentifier object_id_1,
                                   VideoObjectIdentifier object_id_2)
```

For each i-th access unit in the input media stream, the function makes a copy of this i-th access unit. Then, the function sets the position of the video object samples that belong to the video object identified by the second video object identifier below the video object samples belonging to the object identified by the first video object identifier in this copied access unit. This positioning is done in such a way that the left boundaries of both video object samples are aligned. Lastly, the copied access unit is appended to the output media stream as a new access unit.

7.2.6.2 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 neighbors.

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). 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.



a) Input media stream **b) Output elementary stream**

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

7.3 Slice-based instantiation for ISO/IEC 23008-2 high efficiency video coding (HEVC)

7.3.1 General

The high efficiency video coding (HEVC) is published under ISO/IEC 23008-2.

Table 2 provides the bindings of VDI concepts with the concepts defined in ISO/IEC 23008-2.

Table 2 — Correspondence between VDI concepts and HEVC concrete entities

Concept	HEVC definitions
ElementaryStream	bitstream
AccessUnit	access unit
VideoObjectIdentifier	`slice_segment_address`
VideoObjectSample	slice segment

In the remainder of the subclause 7.3, an HEVC elementary stream shall be a compliant bitstream according to ISO/IEC 23008-2.

7.3.2 Media and elementary stream constraints

7.3.2.1 General media stream constraints

A HEVC media stream used as an instantiation of the media stream in subclause 7.2 shall obey the following rules:

— There shall be exactly on slice per tile and exactly one tile per slice.

— The tiling grid shall be constant for each entire coded video sequence.

— Each tile shall be motion-constrained as specified in the semantics of the temporal motion-constrained tile sets SEI message of ISO/IEC 23008-2.

— `dependent_slice_segments_enabled_flag` should be absent or, if present, equal to 0.

All the active SPSs of the HEVC input media stream shall be constrained as follows:

— `general_progressive_source_flag` shall be equal to 1.

— `general_frame_only_constraint_flag` shall be equal to 1.

— `general_interlaced_source_flag` shall be equal to 0.

— `init_qp_minus26` in the PPS shall be set to the same value across all HEVC input media streams.

— The reference picture set (RPS) shall be the same across all HEVC input media streams.

7.4 Layer-based instantiation for ISO/IEC 23090-3 versatile video coding (VVC)

7.4.1 General

The versatile video coding (VVC) is published under ISO/IEC 23090-3.

Table 3 provides the bindings of VDI concepts with the concepts defined in ISO/IEC 23090-3.

Table 3 — Correspondence between VDI concepts and VVC concrete entities

Concept	VVC definitions
ElementaryStream	bitstream
AccessUnit	access unit
VideoObjectIdentifier	`nuh_layer_id`
VideoObjectSample	picture unit

In the remainder of the subclause 7.4, an VVC elementary stream shall be a compliant bitstream according to ISO/IEC 23090-3 and the independent layer info SEI message shall be defined as specified in Annex C.

7.4.2 Media and elementary stream constraints

7.4.2.1 General media stream constraints

A VVC media stream used as an instantiation of the media stream in subclause 7.2 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 0.

7.4.2.2 Media and elementary stream constraints for input formatting functions

7.4.2.2.1 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 subclause 7.4.2.1:

— 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:

— 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.

7.4.2.2.2 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 subclause 7.4.2.1:

— 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 subclause 7.4.2.1:

— 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.

7.4.2.2.3 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 subclause 7.4.2.1:

— 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 subclause 7.4.2.1:

— 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.

7.4.2.2.4 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 subclause 7.4.2.1:

— 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 subclause 7.4.2.1:

— 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.

7.5 Slice-based instantiation for ISO/IEC 23094-1 essential video coding (EVC)

7.5.1 General

The essential video coding (EVC) is published under ISO/IEC 23094-1.

Table 4 provides the bindings of VDI concepts with the concepts specified in ISO/IEC 23094-1.

Table 4 — Correspondence between VDI concepts and EVC concrete entities

Concept	EVC definitions
ElementaryStream	bitstream
AccessUnit	access unit
VideoObjectIdentifier	the smallest value of the ID of the tiles in a slice
VideoObjectSample	slice

In the remainder of the subclause 7.4, an EVC elementary stream shall be a compliant bitstream according to ISO/IEC 23094-1.

7.5.2 Media and elementary streams constraints

7.5.2.1 General media stream constraints

An EVC media stream used as an instantiation of the media stream in subclause 7.2 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 that are different.

7.5.2.2 Media and elementary stream constraints for input formatting functions

7.5.2.2.1 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.

7.5.2.2.2 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.

7.5.2.2.3 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.

7.5.2.2.4 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.

Annex A
(normative)

Control interface IDL definition

The control interface to the video decoding engine is specified using the IDL syntax in ISO/IEC 19516.

The control interface is available at https://standards.iso.org/iso-iec/23090/-13/ed-1/en/.

Annex B
(informative)

OpenMAX IL VDI extension header

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

The source code of the extension is available at https://standards.iso.org/iso-iec/23090/-13/ed-1/en/.

Annex C
(normative)

Supplemental enhancement information (SEI) syntax and semantics

C.1 VDI SEI envelope

C.1.1 General

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

The VDI SEI envelope is registered as a SEI payload in ISO/IEC 23090-3.

C.1.2 VDI SEI envelope syntax

Table C.1 defines the syntax of the VDI SEI envelope.

Table C.1 — Syntax of VDI SEI envelope

Syntax	Size	Type
vdi_sei_envelope(payloadSize) {		
vdi_sub_type	8	unsigned integer
if(vdi_sub_type == 0)		
independent_layer_info(payloadSize - 1)		
else		
reserved_message(payloadSize - 1)		
}		

C.1.3 VDI SEI envelope semantics

vdi_sub_type indicates the payload type carried in the VDI SEI envelope.

C.2 Independent layer info SEI message

C.2.1 Independent layer info SEI message syntax

Table C.2 defines the syntax of the independent layer info SEI message.

Table C.2 — Syntax of independent layer info SEI message

Syntax	Size	Type
independent_layer_info(payloadSize) {		
boundary_identifier_north_present_flag	1	bit
if(boundary_identifier_north_present_flag)		
boundary_identifier_north	16	unsigned integer
boundary_identifier_east_present_flag	1	bit
if(boundary_identifier_east_present_flag)		
boundary_identifier_east	16	unsigned integer
boundary_identifier_south_present_flag	1	bit

Table C.2 *(continued)*

Syntax	Size	Type
if(boundary_identifier_south_present_flag)		
boundary_identifier_south	16	unsigned integer
boundary_identifier_west_present_flag	1	bit
if(boundary_identifier_west_present_flag)		
boundary_identifier_west	16	unsigned integer
}		

C.2.2 Independent layer info SEI message semantics

The independent layer info SEI message provides the spatial alignment of the different independent layers present in a bitstream by epxressing the relative positioning between these layers using matching boundary identifiers. In a multi-layer bitstream, there shall be at most two occurrences of a given boundary identier creating a pair of matching boudary identifier. In other words, a layer has at most one neighbor per boundary.

This SEI message may be extracted by the VDE and used for the output formatting function to correctly place the decoded pictures from each layer in the final output picture. In this case, the decoder instance may ignore this SEI message if present. Alternatively, the decoder instance may be capable of decoding a multi-layer bitstream as well as parsing the SEI message in which case no output formatting function is needed.

`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 C.1 illustrates the where the bounaries lie on the decoded picture associated with a layer.

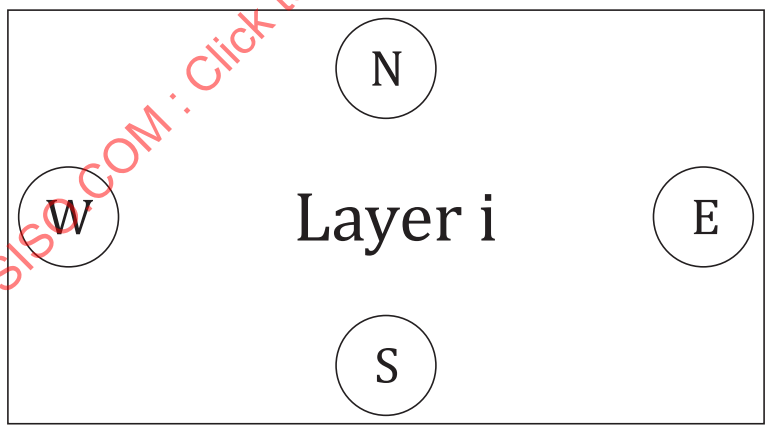


Figure C.1 — Representation of where the boundaries are on the layers

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 east/west boundary.

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.

C.2.3 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, calculate the horizontal, XPos, and vertical, YPos, positions of the top-left corner of each cropped decoded picture per layer in the final output picture. An example step sequence to calculate XPos and YPos is as follows:
    - For each layer:
      - Parse the list of cropped picture size and the boundary identifier.
      - Identify the north, east, south, west neighbouring layers.
    - Place each layer in a grid, with each grid cell corresponding to a north, east, south, west value for each layer that matches the corresponding neighboring value, respectively south, west, north, east.
    - The value XPos for each layer corresponds to the sum of the widths of each layer in the same row of the grid, left of the current layer. The value YPos for each layer is corresponds to the sum of the heights of each layer in the same column of the grid, above the current layer.
  - Initialize a picture buffer of size FinalWidth of width and FinalHeight of height for the final output picture where FinalWidth and FinalHeight are the width and height of the final output picture when all the layers are concatenated according to the defined graph.
  - 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.

If at the end of this process, the combination of all the decoded pictures does not provide decoded sample values for all the samples of the final output picture, the implementation determines the value to be used for these unused samples.

NOTE Since the further process only extracts the decoded samples from the final output picture, the sample value of the unused samples is not relevant from a normative perspective.

C.3 Examples of video object positioning

C.3.1 Appending

This example appends layer 0 to layer 1. Table C.3 gives the properties of each layer.

Table C.3 — Properties of layer 0 and layer 1

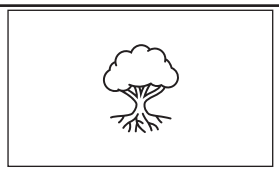
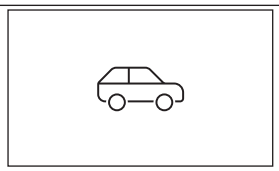
Sequence	Layer	Resolution	Boundary identifiers (N, E, S, W)
	0	416 x 240	0 1 0 0
	1	416 x 240	0 0 0 1

Figure C.2 depicts the signaled connected map.

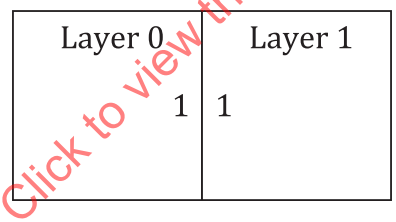
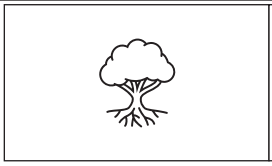
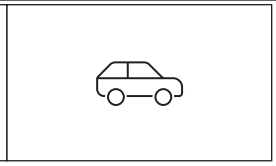


Figure C.2 — Connected map of layer 0 and layer 1

Based on this configuration, Table C.4 presents the properties of the final output pictures.

Table C.4 — Properties of the final output pictures

Resolution	
832 x 240	
Output sequence	
	

C.3.2 Appending and stacking

This example appends layer 0 next to layer 1 and stacks layer 0 on top of layer 2. Table C.5 gives the properties of each layer.

Table C.5 — Properties of layer 0, layer 1 and layer 2

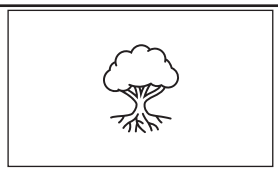
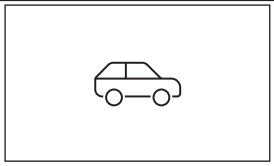
Sequence	Layer	Resolution	Boundary identifiers (N, E, S, W)
	0	416 x 240	0 1 2 0
	1	416 x 240	0 0 0 1
	2	832x480	2 0 0 0

Figure C.3 depicts the signaled connected map.

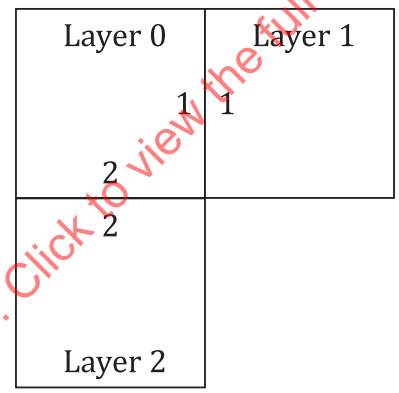
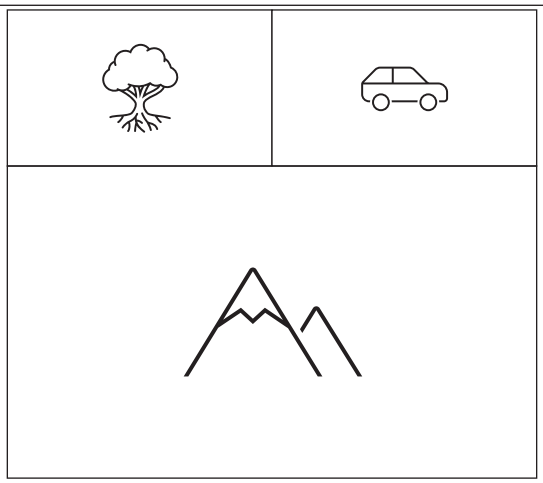


Figure C.3 — Connected map of layer 0 and layer 1

Based on this configuration, Table C.6 presents the properties of the final output pictures.

Table C.6 — Properties of the final output pictures

Resolution
832 x 720
Output sequence


Annex D
(informative)

Example implementations of input formatting operations

D.1 General

The operations defined in subclause 7.2 as well as the associated input and output constraints provide the building blocks for the implementations of the input formatting function. The way a certain implementation converts the media streams to elementary streams based on the requested decoded sequences configuration is informative and left for optimization by the implementor as long as the output elementary streams meet the requirements of the elementary stream interface.

D.2 Creating a 2-by-2 video mosaic via application control

In this example, the goal is to take four video objects as input in four different media streams and obtain as output of the VDE one decoded sequence containing the four decoded video objects arranged in a 2 by 2 grid in each picture of the decoded sequence. We assume that the application is able to communicate to the VDE via an external communication channel to express how the media stream needs to be arranged in the decoded sequences. For the purpose of the illustration, consider that the four video objects have the same video resolution. Figure D.1 depicts the input situation at the media stream interface, input of the VDE and the intended situation at the output of the VDE.

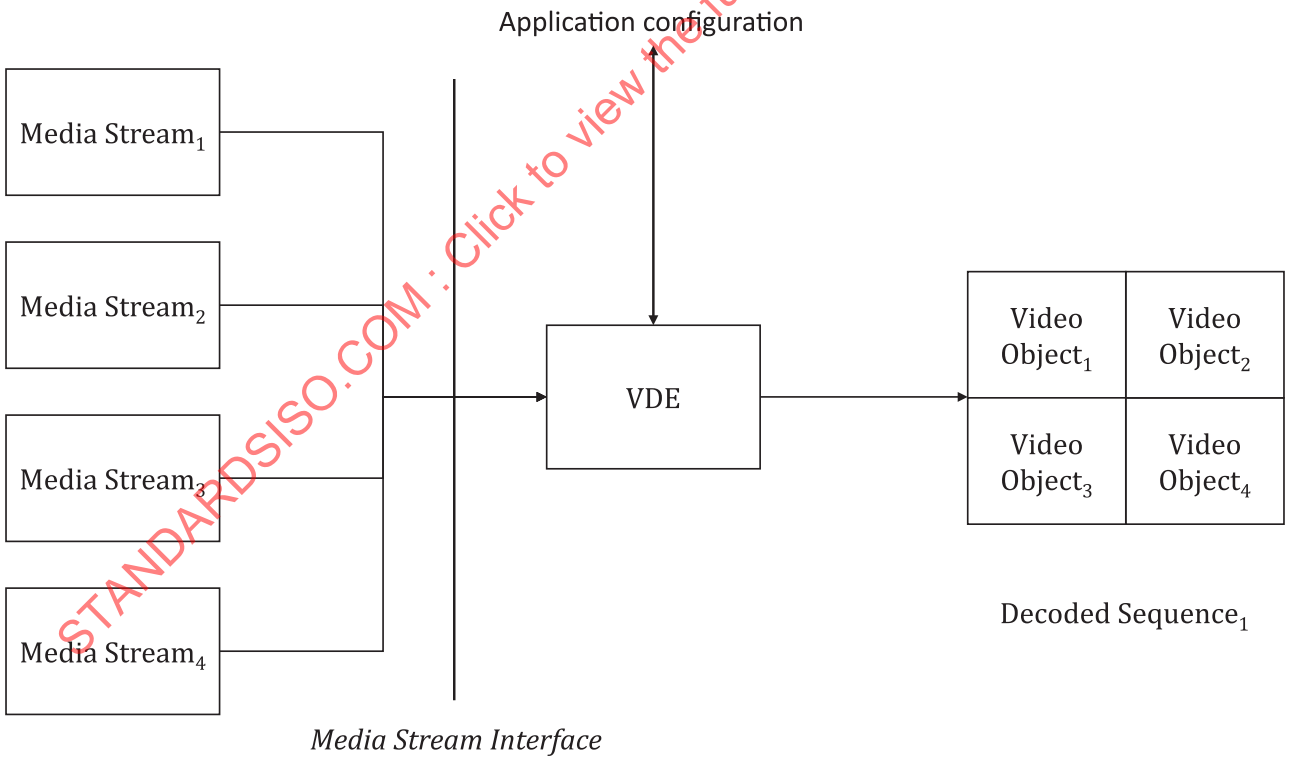


Figure D.1 — Mosaicking of 2-by-2 video objects with application-based VDE control

This document specifies that the output of the input formatting function is the elementary stream interface. That is, the input formatting function needs to output data streams which are elementary streams. Whether it should be one elementary stream or multiple elementary streams is implementation and platform

dependent. For this example, assume two variants. The first one is to have one video decoder per media stream, the second one is to have one single decoder for the four media streams.

In the case of one video decoder per media stream, the input formatting function is effectively an identity operation, i.e. each media stream is passed on to one decoder instance. That also means that each media stream is in this case also an elementary stream which is allowed by the definition of a media stream. Then the VDE runs those four decoder instances in parallel, collects each output picture and assemble them into the 2-by-2 arrangement for each set of temporally collocated decoded pictures as shown on Figure D.2.

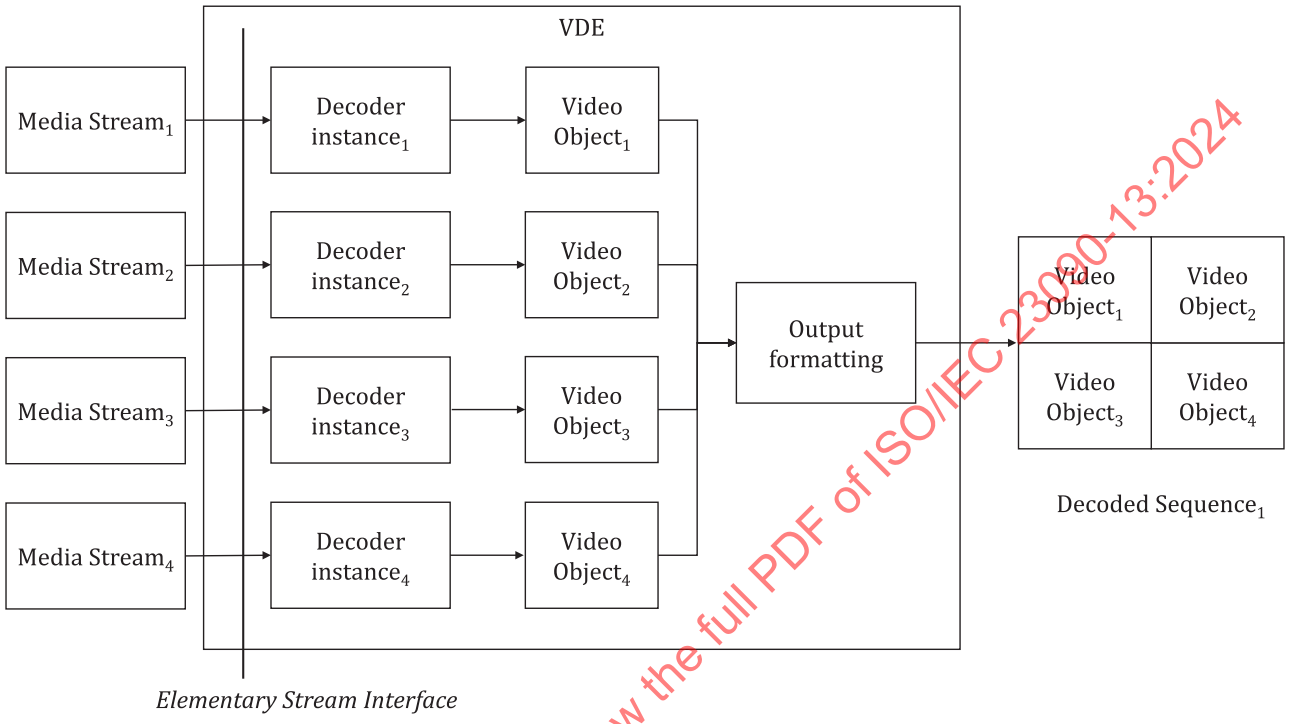


Figure D.2 — Mosaicking of 2-by-2 video objects with four decoder instances

In the case of one video decoder for the four media streams, the input formatting function takes care of creating a single elementary stream out of the four input media streams. From that point onwards, the VDE runs a conventional pipeline with a single decoder and output the decoded picture from the decoder instance without the need of further processing before the output of the VDE. This case is depicted in Figure D.3.