FINAL
DRAFT

INTERNATIONAL
STANDARD

**ISO/IEC
FDIS
23090-13**

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

Secretariat: **JISC**

Voting begins on:
2023-10-19

Voting terminates on:
2023-12-14

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 FDIS 23090-13:2023(E)

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

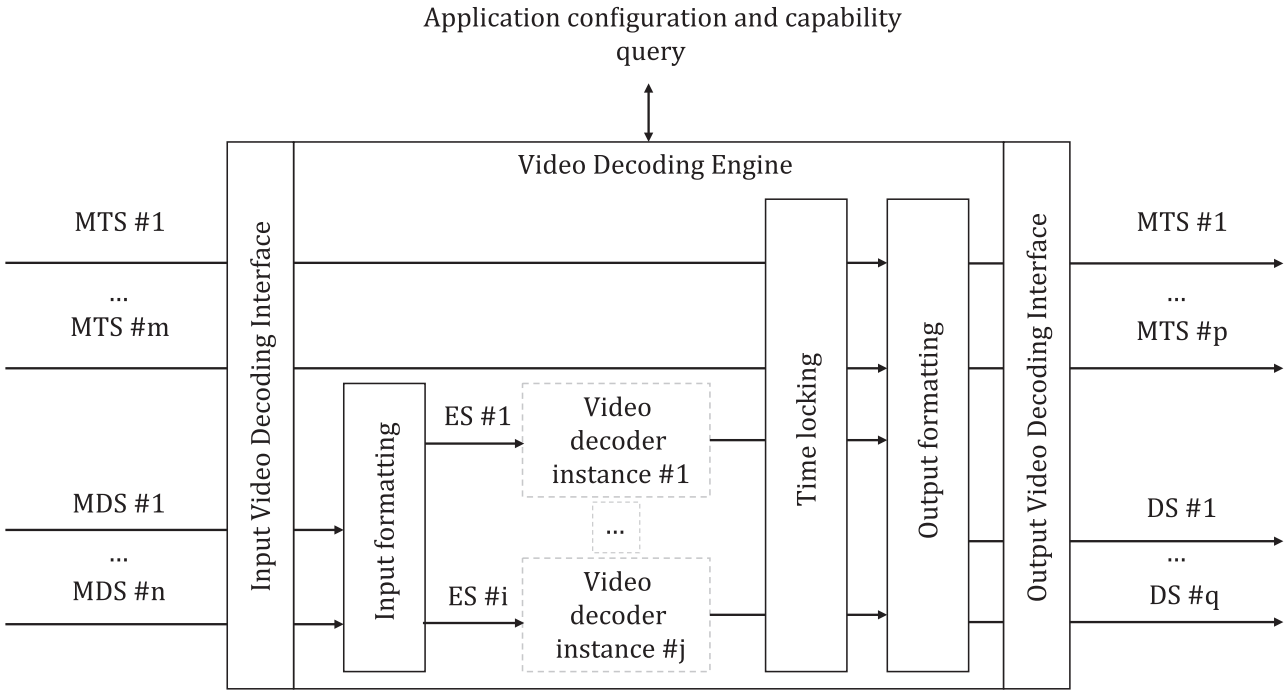
API	application programming interface
ES	elementary stream
I	video object identifier
IDL	interface definition language
IVDI	input video decoding interface
MDS	media stream
NAL	network abstraction layer
OLS	output layer set
OVDI	output video decoding interface
PPS	picture parameter set
SEI	supplemental 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.

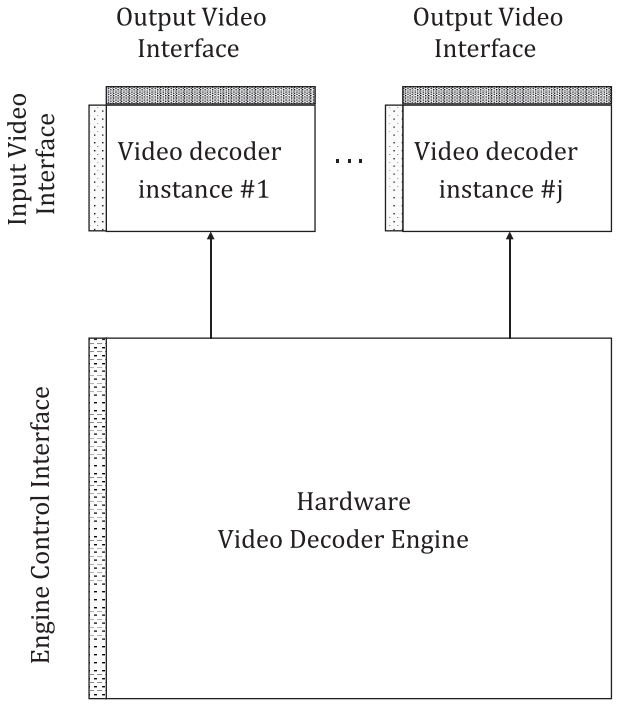


Figure 2 — Example relationship between video decoder instances and video decoder hardware engine

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.

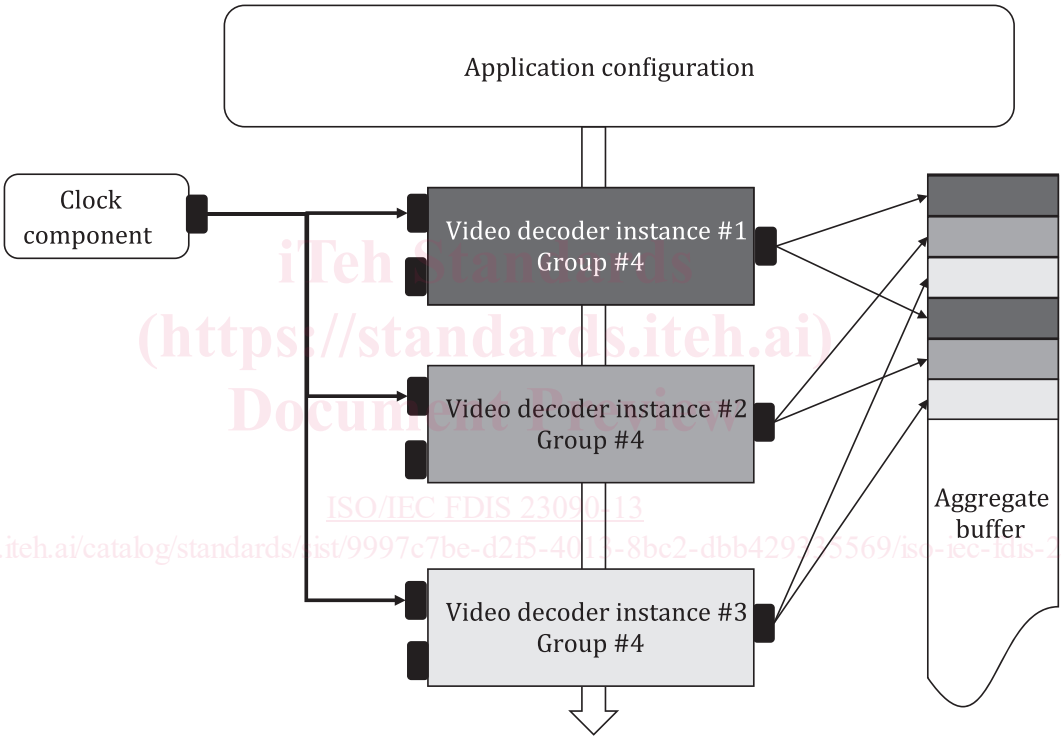


Figure 3 — Example instantiation using VDI

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.