
**Information technology — General
video coding —**

**Part 1:
Essential video coding**

*Technologies de l'information — Codage vidéo général —
Partie 1: Codage vidéo essentiel*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 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
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents	Page
Foreword	vi
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Abbreviated terms	11
5 Conventions	12
5.1 General.....	12
5.2 Arithmetic operators	13
5.3 Logical operators.....	13
5.4 Relational operators	13
5.5 Bit-wise operators	13
5.6 Assignment operators	14
5.7 Range notation	14
5.8 Mathematical functions	14
5.9 Order of operation precedence	16
5.10 Variables, syntax elements and tables.....	16
5.11 Text description of logical operations	18
5.12 Processes	19
6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships	19
6.1 Bitstream formats	19
6.2 Source, decoded and output picture formats	20
6.3 Partitioning of pictures, slices, tiles, and CTUs	22
6.3.1 Partitioning of pictures into slices and tiles.....	22
6.3.2 Spatial or component-wise partitionings.....	23
6.4 Availability processes.....	24
6.4.1 Derivation process for neighbouring block availability.....	24
6.4.2 Derivation process for left and right neighbouring blocks availabilities....	24
6.4.3 Derivation process for neighbouring block motion vector candidate availability.....	25
6.4.4 Derivation process for ALF neighbouring block availability.....	25
6.5 Scanning processes.....	26
6.5.1 CTB raster and tile scanning process.....	26
6.5.2 Zig-zag scan order 1D array initialization process.....	28
6.5.3 Inverse scan order 1D array initialization process	29
7 Syntax and semantics	29
7.1 Method of specifying syntax in tabular form	29
7.2 Specification of syntax functions and descriptors.....	31
7.3 Syntax in tabular form.....	32
7.3.1 NAL unit syntax.....	32
7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax	33
7.3.3 Supplemental enhancement information message syntax	38
7.3.4 Slice header syntax	39
7.3.5 Adaptive loop filter data syntax	41
7.3.6 DRA data syntax.....	42

7.3.7	Reference picture list structure syntax.....	43
7.3.8	Slice data syntax	43
7.4	Semantics.....	56
7.4.1	General.....	56
7.4.2	NAL unit semantics.....	56
7.4.3	Raw byte sequence payloads, trailing bits and byte alignment semantics	60
7.4.4	Supplemental enhancement information message semantics	73
7.4.5	Slice header semantics	74
7.4.6	Adaptive loop filter data semantics	79
7.4.7	DRA data semantics.....	84
7.4.8	Reference picture list structure semantics.....	86
7.4.9	Slice data semantics	88
8	Decoding process	105
8.1	General decoding process.....	105
8.2	NAL unit decoding process	105
8.3	Slice decoding process	105
8.3.1	Decoding process for picture order count.....	105
8.3.2	Decoding process for reference picture lists construction.....	107
8.3.3	Decoding process for reference picture marking	111
8.3.4	Decoding process for collocated picture	112
8.4	Decoding process for coding units coded in intra prediction mode	112
8.4.1	General.....	112
8.4.2	Derivation process for luma intra prediction mode	114
8.4.3	Derivation process for chroma intra prediction mode.....	124
8.4.4	Decoding process of intra prediction.....	126
8.4.5	Decoding process for the residual signal.....	141
8.5	Decoding process for coding units coded in inter prediction mode	143
8.5.1	General.....	143
8.5.2	Derivation process for motion vector components and reference indices.....	148
8.5.3	Derivation process for affine motion vector components and reference indices	188
8.5.4	Decoding process for inter prediction samples.....	217
8.5.5	Decoder-side motion vector refinement process	234
8.5.6	Decoding process for the residual signal of coding units coded in inter prediction mode.....	240
8.6	Decoding process for coding units coded in ibc prediction mode.....	246
8.6.1	General.....	246
8.6.2	Derivation process for motion vector components	247
8.6.3	Decoding process for ibc blocks.....	250
8.7	Scaling, transformation and array construction process	251
8.7.1	Derivation process for quantization parameters.....	251
8.7.2	Scaling and transformation process.....	251
8.7.3	Scaling process for transform coefficients.....	252
8.7.4	Transformation process for scaled transform coefficients.....	253
8.7.5	Picture construction process.....	263
8.7.6	Post-reconstruction filter process.....	264
8.8	In-loop filter process	267
8.8.1	General.....	267
8.8.2	Deblocking filter process	268
8.8.3	Advanced deblocking filter process.....	280
8.8.4	Adaptive Loop Filter	293
8.9	DRA process	303

8.9.1	General.....	303
8.9.2	Derivation of samples of output decoded picture by DRA process	303
8.9.3	Inverse mapping process for a luma sample	304
8.9.4	Inverse mapping process for a chroma sample.....	305
8.9.5	Identification of the range index of piecewise function	305
8.9.6	DRA chroma scale value derivaton process	306
8.9.7	Derivation of output chroma DRA parameters.....	306
8.9.8	Derivation of adjusted chroma DRA scales.....	307
9	Parsing process	309
9.1	General.....	309
9.2	Parsing process for 0-th order Exp-Golomb codes	310
9.2.1	General.....	310
9.2.2	Mapping process for signed Exp-Golomb codes.....	311
9.3	CABAC parsing process for slice data	312
9.3.1	General.....	312
9.3.2	Initialization process	312
9.3.3	Binarization process	326
9.3.4	Decoding process flow	333
	Annex A (normative) Profiles, levels and toolsets.....	349
	Annex B (normative) Raw bitstream file storage format	361
	Annex C (normative) Hypothetical reference decoder	362
	Annex D (normative) Supplemental enhancement information.....	374
	Annex E (normative) Video usability information.....	389
	Bibliography	414

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).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC 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) or the IEC list of patent declarations received (see <http://patents.iec.ch>).

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.

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/IEC 23094 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.

Introduction

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — General video coding —

Part 1: Essential video coding

1 Scope

This document specifies a video coding technology known as essential video coding (EVC), which contains syntax format, semantics and an associated decoding process. The decoding process is designed to guarantee that all EVC decoders conform to a specified combination of capabilities known as the profile, level and toolset. Any decoding process that produces identical cropped decoded output pictures to those produced by the described process is considered to be in conformance with the requirements of this document.

This document is designed to cover a wide range of application, including but not limited to digital storage media, television broadcasting and real-time communications.

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 11664-1, *Colorimetry — Part 1: CIE standard colorimetric observers*

3 Terms and definitions

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

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1

access unit

set of *NAL units* (3.53) that are associated with each other according to a specified classification rule, are consecutive in *decoding order* (3.28), and contain exactly one *coded picture* (3.16)

3.2

adaptive loop filter

ALF

filtering process that is applied as part of the *decoding process* (3.29) and is controlled by *parameters* (3.58) conveyed in an *APS* (3.4)

3.3

ALF APS

APS (3.4) that controls the *ALF* (3.2) process

3.4

adaptation parameter set

APS

syntax structure (3.87) containing *syntax elements* (3.86) that apply to zero or more *slices* (3.81) or *pictures* (3.60) as determined by zero or more *syntax elements* (3.86) found in *slice headers* (3.82)

3.5

bi-predictive slice

B slice

slice (3.81) that is decoded using *intra prediction* (3.43) or using *inter prediction* (3.40) with at most two *motion vectors* (3.52) and *reference indices* (3.74) to predict the sample values of each *block* (3.11)

3.6

bin

one bit of a *bin string* (3.7)

3.7

bin string

intermediate binary representation of values of *syntax elements* (3.86) from the *binarization* (3.8) of the *syntax element* (3.86)

3.8

binarization

set of *bin strings* (3.7) for all possible values of a *syntax element* (3.86)

3.9

binarization process

unique mapping process of all possible values of a *syntax elements* (3.86) from the *binarization* (3.8) of the *syntax element* (3.86)

3.10

bitstream

sequence of bits, in the form of a *NAL unit stream* (3.54) or a *raw bitstream* (3.71), that forms the representation of *coded pictures* (3.16) and associated data forming one or more *coded video sequences* (3.19)

3.11

block

MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients* (3.93)

3.12

buffering period

set of *access units* (3.1) starting with an *access unit* (3.1) that contains a buffering period supplemental enhancement information (SEI) message and containing all subsequent *access units* (3.1) in *decoding order* (3.28) up to but not including the next *access unit* (3.1) (when present) that contains a buffering period SEI message

3.13

byte

sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively

3.14**byte-aligned**

position in a *bitstream* (3.10) where the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream* (3.10)

Note 1 to entry: A bit or *byte* (3.13) or *syntax element* (3.86) is said to be byte-aligned when the position at which it appears in a *bitstream* (3.10) is byte-aligned.

3.15**chroma**

sample array or single sample representing one of the two colour difference signals related to the primary colours, represented by the symbols Cb and Cr

Note 1 to entry: The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that are often associated with the term chrominance.

3.16**coded picture**

coded representation (3.18) of a *picture* (3.60) containing all *CTUs* (3.22) of the *picture* (3.60)

3.17**coded picture buffer****CPB**

first-in first-out buffer containing *access units* (3.1) in *decoding order* (3.28)

Note 1 to entry: Specified in the *hypothetical reference decoder* (3.36) in Annex C.

3.18**coded representation**

data element as represented in its coded form

3.19**coded video sequence****CVS**

sequence of *access units* (3.1) that consists, in *decoding order* (3.28), of an *IDR access unit* (3.38), followed by zero or more *access units* (3.1) that are not *IDR access units* (3.38), including all subsequent *access units* (3.1) up to but not including any subsequent *access unit* (3.1) that is an *IDR access unit* (3.38)

3.20**coding block****CB**

MxN *block* (3.11) of samples for some values of M and N such that the division of a *CTB* (3.21) into coding blocks is a *partitioning* (3.59)

3.21**coding tree block****CTB**

NxN *block* (3.11) of samples for some value of N such that the division of a *component* (3.24) into coding tree blocks is a *partitioning* (3.59)

3.22**coding tree unit****CTU**

CTB (3.21) of *luma* (3.51) samples, two corresponding *CTBs* (3.21) of *chroma* (3.15) samples of a *picture* (3.60) that has three sample arrays, or a *CTB* of samples of a monochrome *picture* (3.60) or a *picture*

(3.60) that is coded using three separate colour planes and *syntax structures* (3.87) used to code the samples

3.23

coding unit

CU

coding block (3.20) of *luma* (3.51) samples, two corresponding *coding blocks* (3.20) of *chroma* (3.15) samples of a *picture* (3.60) that has three sample arrays, or a *coding block* (3.20) of samples of a monochrome *picture* (3.60) or a *picture* (3.60) that is coded using three separate colour planes and *syntax structures* (3.87) used to code the samples

3.24

component

array or single sample from one of the three arrays (*luma* (3.51) and two *chroma* (3.15)) that compose a *picture* (3.60) in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* (3.60) in monochrome format

3.25

decoded picture

derived by decoding a *coded picture* (3.16)

3.26

decoded picture buffer

DPB

buffer holding *decoded pictures* (3.25) for reference, output reordering, or output delay

Note 1 to entry: Specified for the *hypothetical reference decoder* (3.36) in Annex C.

3.27

decoder

embodiment of a *decoding process* (3.29)

3.28

decoding order

order in which *syntax elements* (3.86) are processed by the *decoding process* (3.29)

3.29

decoding process

process specified that reads a *bitstream* (3.10) and derives *decoded pictures* (3.25) from it

3.30

dynamic range adjustment

DRA

mapping process that is applied to *decoded picture* (3.25) prior to cropping and output as part of the *decoding process* (3.29) and is controlled by parameters conveyed in an *APS* (3.4)

3.31

DRA APS

APS (3.4) that controls the *DRA* (3.30) process

3.32**decoder under test****DUT**

decoder (3.27) that is tested for conformance to this document

Note 1 to entry: A *decoder* (3.27) is tested by operating the *hypothetical stream scheduler* (3.37) to deliver a conforming *bitstream* (3.10) to the *decoder* (3.27) and to the *hypothetical reference decoder* (3.36) and comparing the values and timing of the output of the two *decoders* (3.27).

3.33**encoder**

embodiment of an *encoding process* (3.34)

3.34**encoding process**

process that produces a *bitstream* (3.10) conforming to this document

3.35**flag**

variable or single-bit *syntax element* (3.86) that can take one of the two possible values: 0 and 1

3.36**hypothetical reference decoder****HRD**

hypothetical *decoder* (3.27) model that specifies constraints on the variability of conforming *NAL unit streams* (3.54) or conforming *raw bitstreams* (3.10) produced by an encoding process

3.37**hypothetical stream scheduler****HSS**

hypothetical delivery mechanism for the timing and data flow of the input of a *bitstream* (3.10) into the *hypothetical reference decoder* (3.36)

Note 1 to entry: The HSS is used for checking the conformance of a *bitstream* (3.10) or a *decoder* (3.36).

3.38**IDR access unit**

access unit in which the *coded picture* is an *IDR picture*

3.39**IDR picture**

coded picture (3.16) for which each *VCL NAL unit* (3.96) has *NalUnitType* equal to *IDR_NUT*

3.40**inter prediction**

prediction (3.63) derived in a manner that is dependent on data elements (e.g., sample values or motion vectors) of one or more *reference pictures* (3.75)

Note 1 to entry: A *prediction* (3.63) from a *reference picture* (3.75) that is the current *picture* (3.60) itself is also inter prediction.

3.41

intra block copy

IBC

prediction (3.63) derived in a manner that is dependent on data elements (e.g., sample values or block vectors) of the same decoded *slice* (3.81) without referring to a *reference picture* (3.75)

3.42

intra coding

coding of a *coding block* (3.20), *slice* (3.81), or *picture* (3.60) that uses *intra prediction* (3.43)

3.43

intra prediction

prediction (3.63) derived from only data elements (e.g., sample values) of the same decoded *slice* (3.81) without referring to a *reference picture* (3.75)

3.44

intra slice

I slice

slice (3.81) that is decoded using *intra prediction* (3.43) only

3.45

level

defined set of constraints on the values that may be taken by the *syntax elements* (3.86) and variables of this document, or the value of a *transform coefficient* (3.93) prior to scaling

Note 1 to entry: The same set of levels is defined for all *profiles* (3.67), with most aspects of the definition of each level being in common across different *profiles* (3.67). Individual implementations can, within the specified constraints, support a different level for each supported *profile* (3.67).

3.46

list 0 motion vector

motion vector (3.52) associated with a *reference index* (3.74) pointing into a *reference picture list 0* (3.77)

3.47

list 0 prediction

inter prediction (3.40) of the content of a *slice* (3.81) using a *reference index* (3.74) pointing into a *reference picture list 0* (3.77)

3.48

list 1 motion vector

motion vector (3.52) associated with a *reference index* (3.74) pointing into a *reference picture list 1* (3.78)

3.49

list 1 prediction

inter prediction (3.40) of the content of a *slice* (3.81) using a *reference index* (3.74) pointing into a *reference picture list 1* (3.78)

3.50

long-term reference picture

LTRP

picture (3.60) that is marked as "used for long-term reference"

3.51**luma**

sample array or single sample is representing the monochrome signal related to the primary colours, represented by the symbol or subscript Y or L

Note 1 to entry: The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that are often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

3.52**motion vector**

two-dimensional vector used for *inter prediction* (3.40) that provides an offset from the coordinates in the *decoded picture* (3.25) to the coordinates in a *reference picture* (3.75)

3.53**NAL unit**

syntax structure (3.87) containing an indication of the type of data to follow and *bytes* (3.13) containing that data in the form of an *RBSP* (3.72) interspersed as necessary

3.54**NAL unit stream**

sequence of *NAL units* (3.53)

3.55**non-IDR picture**

coded picture (3.16) that is not an *IDR picture* (3.39)

3.56**non-VCL NAL unit**

NAL unit (3.53) that is not a *VCL NAL unit* (3.96)

3.57**output order**

order in which the *decoded pictures* (3.25) are output from the *decoded picture buffer* (3.26) [for the *decoded pictures* (3.25) that are to be output from the *decoded picture buffer* (3.26)]

3.58**parameter**

syntax element (3.86) of an *SPS* (3.79), *PPS* (3.61) or *APS* (3.4)

3.59**partitioning**

division of a set into subsets such that each element of the set is in exactly one of the subsets

3.60**picture**

array of *luma* (3.51) samples in monochrome format or an array of *luma* (3.51) samples and two corresponding arrays of *chroma* (3.15) samples in 4:2:0, 4:2:2, and 4:4:4 colour format

Note 1 to entry: A picture can be either a frame or a field. However, in one *CVS* (3.19), either all pictures are frames or all pictures are fields.

3.61

picture parameter set

PPS

syntax structure (3.87) containing *syntax elements* (3.86) that apply to zero or more entire *coded pictures* (3.16) as determined by a *syntax element* (3.86) found in each *slice header* (3.82)

3.62

picture order count

POC

variable that is associated with each *picture* (3.60), uniquely identifies the associated *picture* (3.60) among all *pictures* (3.60) in the *CVS* (3.19), and, when the associated *picture* (3.60) is to be output from the *decoded picture buffer* (3.26), indicates the position of the associated *picture* (3.60) in *output order* (3.57) relative to the *output order* (3.57) positions of the other *pictures* (3.60) in the same *CVS* (3.19) that are to be output from the *decoded picture buffer* (3.26)

3.63

prediction

embodiment of the *prediction process* (3.63)

3.64

prediction process

use of a *predictor* (3.66) to provide an estimate of the data element [e.g., sample value or *motion vector* (3.52)] currently being decoded

3.65

predictive slice

P slice

slice (3.81) that is decoded using *intra prediction* (3.43) or using *inter prediction* (3.40) with at most one *motion vector* (3.52) and *reference index* (3.74) to predict the sample values of each *block* (3.11)

3.66

predictor

combination of specified values or previously decoded data elements [e.g., sample value or *motion vector* (3.52)] used in the *decoding process* (3.29) of subsequent data elements

3.67

profile

specified subset of the syntax of this document

3.68

quantization parameter

QP

variable used by the *decoding process* (3.29) for scaling of *transform coefficient* (3.93) levels

3.69

random access

act of starting the decoding process for a *bitstream* (3.10) at a point other than the beginning of the stream

3.70

raster scan

mapping of a rectangular two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the first top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc., rows of the pattern (going down) each scanned from left to right

3.71**raw bitstream**

encapsulation of a *NAL unit stream* (3.54) containing *NAL unit* (3.53) length field and *NAL units* (3.53)

Note 1 to entry: Specified in Annex A.

3.72**raw byte sequence payload****RBSP**

syntax structure (3.87) containing an integer number of *bytes* (3.13) that is encapsulated in a *NAL unit* (3.53) and that is either empty or has the form of a *string of data bits* (3.85) containing *syntax elements* (3.86) followed by an *RBSP stop bit* (3.73) and zero or more subsequent bits equal to 0

3.73**raw byte sequence payload stop bit****RBSP stop bit**

bit equal to 1 present within a *raw byte sequence payload* (3.72) after a *string of data bits* (3.85), for which the location of the end within an *RBSP* (3.72) can be identified by searching from the end of the *RBSP* (3.72) for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP* (3.72)

3.74**reference index**

index into a *reference picture list* (3.76)

3.75**reference picture**

picture (3.60) that is a *short-term reference picture* (3.80) or *long-term reference picture* (3.50)

Note 1 to entry: A reference picture contains samples that can be used for *inter prediction* (3.40) in the *decoding process* (3.29) of subsequent *pictures* (3.60) in *decoding order* (3.28).

3.76**reference picture list**

list of *reference pictures* (3.80) that is used for *inter prediction* (3.40) of a *P slice* (3.65) or *B slice* (3.5)

Note 1 to entry: For the *decoding process* (3.29) of a *P slice* (3.65), there is one reference picture list – *reference picture list 0* (3.77). For the *decoding process* (3.28) of a *B slice* (3.5), there are two reference picture lists – *reference picture list 0* (3.77) and *reference picture list 1* (3.78).

3.77**reference picture list 0**

reference picture list (3.76) used for *inter prediction* of a *P slice* (3.65) or the first *reference picture list* (3.76) used for *inter prediction* (3.40) of a *B slice* (3.5)

3.78**reference picture list 1**

second *reference picture list* (3.76) used for *inter prediction* (3.40) of a *B slice* (3.5)

3.79**sequence parameter set****SPS**

syntax structure (3.87) containing *syntax elements* (3.86) that apply to zero or more entire *CVSs* as determined by the content of a *syntax element* (3.86) found in the *PPS* (3.61) referred to by a *syntax element* (3.86) found in each *slice header* (3.82)

3.80

short-term reference picture

STRP

picture (3.60) that is marked as "used for short-term reference"

3.81

slice

integer number of *tiles* (3.88) of a *picture* (3.60) in the *tile scan* (3.91) of the *picture* (3.60) and that are exclusively contained in a single *NAL unit* (3.53)

3.82

slice header

part of a coded *slice* (3.81) containing the data elements pertaining to the first or all *tiles* (3.88) represented in the *slice* (3.81)

3.83

source

term used to describe the video material or some of its attributes before *encoding process* (3.34)

3.84

split unit

SU

MxN *block* (3.11) of samples for some values and M and N such that the division of a *component* (3.24) into four quad *blocks* (3.11), or three horizontal *blocks* (3.11) or two horizontal *blocks* (3.11), or three vertical *blocks* (3.11) or two vertical *blocks* (3.11)

3.85

string of data bits

SODB

sequence of some number of bits representing *syntax elements* (3.86) present within a *raw byte sequence payload* (3.72) prior to the *raw byte sequence payload stop bit* (3.73), where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit

3.86

syntax element

element of data represented in the *bitstream* (3.10)

3.87

syntax structure

zero or more *syntax elements* (3.86) present together in the *bitstream* (3.10) in a specified order

3.88

tile

rectangular region of *CTUs* (3.22) within a particular *tile column* (3.89) and a particular *tile row* (3.90) in a *picture* (3.60)

3.89

tile column

rectangular region of *CTUs* (3.22) having a height equal to the height of the *picture* (3.60) and width specified by *syntax elements* (3.86) in the *PPS* (3.61)

3.90**tile row**

rectangular region of *CTUs* (3.22) having a height specified by *syntax elements* (3.86) in the *PPS* (3.61) and a width equal to the width of the *picture* (3.60)

3.91**tile scan**

specific sequential ordering of *CTUs* (3.22) *partitioning* (3.59) a *picture* (3.60) in which the *CTUs* (3.22) are ordered consecutively in *CTU* (3.22) *raster scan* (3.70) in a *tile* (3.88) whereas *tiles* (3.88) in a *picture* (3.60) are ordered consecutively in a *raster scan* (3.70) of the *tiles* (3.88) of the *picture* (3.60)

3.92**transform block****TB**

rectangular $M \times N$ *block* (3.11) of samples resulting from a transform in the *decoding process* (3.29)

3.93**transform coefficient**

scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional frequency index in a transform in the *decoding process* (3.29)

3.94**transform unit****TU**

transform block (3.94) of *luma* (3.51) samples and two corresponding *transform blocks* (3.94) of *chroma* (3.15) samples of a *picture* (3.60)

3.95**tree**

tree is a finite set of nodes with a unique root node

3.96**VCL NAL unit**

collectively, coded *slice* (3.81) *NAL units* (3.53) and the subset of *NAL units* (3.53) that have reserved values of *NalUnitType* that are classified as VCL NAL units in this document

4 Abbreviated terms

ATS	adaptive transform selection
B	bi-predictive
CABAC	context-based adaptive binary arithmetic coding
CBR	constant bit rate
CIE	International Commission on Illumination (Commission Internationale de l'Eclairage)
DMVR	decoder-side motion vector refinement
EG	exponential-Golomb
EGk	k-th order exponential-Golomb
FCC	Federal Communications Commission (of the United States)
FIR	finite impulse response

FL	fixed length
GOP	group of pictures
HMVP	history-based motion vector prediction
I	intra
IDR	instantaneous decoding refresh
LSB	least significant bit
LPS	least probable symbol
MAC	multiplexed analogue components
MMVD	merge with motion vector difference
MPS	most probable symbol
MSB	most significant bit
MVP	motion vector prediction
NAL	network abstraction layer
NTSC	National Television System Committee (of the United States)
P	predictive
PAL	phase alternating line
RGB	same as GBR
RPL	reference picture list
SAR	sample aspect ratio
SECAM	sequential colour with memory (séquentiel couleur avec mémoire)
SEI	supplemental enhancement information
SMPTE	Society of Motion Picture and Television Engineers
TB	truncated binary
TR	truncated rice
U	unary
UCS	universal coded character set
UTF	UCS transmission format
VBR	variable bit rate
VCL	video coding layer
VUI	video usability information

5 Conventions

5.1 General

NOTE The mathematical operators used in this document are similar to those used in the C programming language. However, the results of integer division and arithmetic shift operations are defined more precisely, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0, e.g., "the first" is equivalent to the 0-th, "the second" is equivalent to the 1-th, etc.

5.2 Arithmetic operators

+	addition
−	subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	multiplication, including matrix multiplication
×	multiplication, including matrix multiplication
x^y	exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation
/	integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1
÷	division in mathematical equations where no truncation or rounding is intended
$\frac{x}{y}$	division in mathematical equations where no truncation or rounding is intended
$\sum_{i=x}^y f(i)$	summation of f(i) with i taking all integer values from x up to and including y
$x \% y$	modulus. Remainder of x divided by y, defined only for integers x and y with $x \geq 0$ and $y > 0$

5.3 Logical operators

$x \ \&\& \ y$	boolean logical "and" of x and y
$x \ \ y$	boolean logical "or" of x and y
!	boolean logical "not"
$x \ ? \ y : z$	if x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the value of z

5.4 Relational operators

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
=	equal to
!=	not equal to

When a relational operator is applied to a syntax element or variable that has been assigned the value "na" (not applicable), the value "na" is treated as a distinct value for the syntax element or variable. The value "na" is considered not to be equal to any other value.

5.5 Bit-wise operators

&	bit-wise "and" When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
---	---

	bit-wise "or" When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0
^	bit-wise "exclusive or" When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
x >> y	arithmetic right shift of a two's complement integer representation of x by y binary digits This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.
x << y	arithmetic left shift of a two's complement integer representation of x by y binary digits This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

5.6 Assignment operators

=	assignment operator
++	increment, i.e., x++ is equivalent to x = x + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation
--	decrement, i.e., x-- is equivalent to x = x - 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation
+=	increment by amount specified, i.e., x += 3 is equivalent to x = x + 3, and x += (-3) is equivalent to x = x + (-3)
-=	decrement by amount specified, i.e., x -= 3 is equivalent to x = x - 3, and x -= (-3) is equivalent to x = x - (-3)

5.7 Range notation

x = y..z	x takes on integer values starting from y to z, inclusive, with x, y, and z being integer numbers and z being greater than y
----------	--

5.8 Mathematical functions

$$\text{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \quad (1)$$

$$\text{Asin}(x) \text{ trigonometric inverse sine function, operating on an argument } x \text{ that is in the range of } -1.0 \text{ to } 1.0, \text{ inclusive, with an output value in the range of } -\pi/2 \text{ to } \pi/2, \text{ inclusive, in units of radians} \quad (2)$$

$$\text{Atan}(x) \text{ trigonometric inverse tangent function, operating on an argument } x, \text{ with an output value in the range of } -\pi/2 \text{ to } \pi/2, \text{ inclusive, in units of radians} \quad (3)$$

$$\text{Atan2}(y, x) = \begin{cases} \text{Atan}\left(\frac{y}{x}\right) & ; \quad x > 0 \\ \text{Atan}\left(\frac{y}{x}\right) + \pi & ; \quad x < 0 \ \&\& \ y \geq 0 \\ \text{Atan}\left(\frac{y}{x}\right) - \pi & ; \quad x < 0 \ \&\& \ y < 0 \\ +\frac{\pi}{2} & ; \quad x = 0 \ \&\& \ y \geq 0 \\ -\frac{\pi}{2} & ; \quad \text{otherwise} \end{cases} \quad (4)$$

$\text{Ceil}(x)$ smallest integer greater than or equal to x (5)

$\text{Clip1}_Y(x) = \text{Clip3}(0, (1 \ll \text{BitDepth}_Y) - 1, x)$ (6)

$\text{Clip1}_C(x) = \text{Clip3}(0, (1 \ll \text{BitDepth}_C) - 1, x)$ (7)

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \quad (8)$$

$\text{Cos}(x)$ trigonometric cosine function operating on an argument x in units of radians (9)

$\text{Floor}(x)$ largest integer less than or equal to x (10)

$$\text{GetCurrMsb}(a, b, c, d) = \begin{cases} c + d & ; \quad b - a \geq d / 2 \\ c - d & ; \quad a - b > d / 2 \\ c & ; \quad \text{otherwise} \end{cases} \quad (11)$$

$\text{Ln}(x)$ natural logarithm of x (the base- e logarithm, where e is the natural logarithm base constant 2.718 281 828...) (12)

$\text{Log2}(x)$ base-2 logarithm of x (13)

$\text{Log10}(x)$ base-10 logarithm of x (14)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases} \quad (15)$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases} \quad (16)$$

$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5)$ (17)

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases} \quad (18)$$

$\text{Sin}(x)$ trigonometric sine function operating on an argument x in units of radians (19)

$\text{Sqrt}(x) = \sqrt{x}$ (20)

$\text{Swap}(x, y) = (y, x)$ (21)

$\text{Tan}(x)$ trigonometric tangent function operating on an argument x in units of radians (22)

5.9 Order of operation precedence

When the order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

- Operations of a higher precedence are evaluated before any operation of a lower precedence.
- Operations of the same precedence are evaluated sequentially from left to right.

Table 1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE For those operators that are also used in the C programming language, the order of precedence used in this document is the same as used in the C programming language.

Table 1 — Operation precedence from highest (top of the table) to lowest (bottom of the table)

operations (with operands x, y, and z)
"x++", "x--"
"!x", "-x" (as a unary prefix operator)
x^y
"x * y", "x × y", "x / y", "x ÷ y", " $\frac{x}{y}$ ", "x % y"
"x + y", "x - y" (as a two-argument operator), " $\sum_{i=x}^y f(i)$ "
"x << y", "x >> y"
"x < y", "x <= y", "x > y", "x >= y"
"x == y", "x != y"
"x & y"
"x y"
"x && y"
"x y"
"x ? y : z"
"x..y"
"x = y", "x += y", "x -= y"

5.10 Variables, syntax elements and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases, the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper

case letters and without any underscore characters. Variables with names starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables with names starting with an upper case letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables with names starting with a lower case letter are only used within the subclause in which they are derived.

In some cases, "mnemonic" names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes "mnemonic" names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper case letter and may contain more upper case letters.

NOTE The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 7.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.8) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as $s[x][y]$ or as s_{yx} . A single column of a matrix may be referred to as a list and denoted by the omission of the row index. Thus, the column of a matrix s at horizontal position x may be referred to as the list $s[x]$.

A specification of values of the entries in rows and columns of an array may be denoted by $\{ \{ \dots \} \{ \dots \} \}$, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix s equal to $\{ \{ 1 \ 6 \} \{ 4 \ 9 \} \}$ specifies that $s[0][0]$ is set equal to 1, $s[1][0]$ is set equal to 6, $s[0][1]$ is set equal to 4, and $s[1][1]$ is set equal to 9.

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value different from zero.

5.11 Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0 )
    statement 0
else if( condition 1 )
    statement 1
...
else /* informative remark on remaining condition */
    statement n
```

may be described in the following manner:

... as follows / ... the following applies:

- If condition 0, statement 0
- Otherwise, if condition 1, statement 1
- ...
- Otherwise (informative remark on remaining condition), statement n

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0a && condition 0b )
    statement 0
else if( condition 1a || condition 1b )
    statement 1
...
else
    statement n
```

may be described in the following manner:

... as follows / ... the following applies:

- If all of the following conditions are true, statement 0:
 - condition 0a
 - condition 0b
- Otherwise, if one or more of the following conditions are true, statement 1:
 - condition 1a
 - condition 1b
- ...
- Otherwise, statement n

In the text, a statement of logical operations as would be described mathematically in the following form:

```

if( condition 0 )
    statement 0
if( condition 1 )
    statement 1

```

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

5.12 Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lower case variable explicitly specified as input. Each process specification has explicitly specified an output. The output is a variable that can either be an upper case variable or a lower case variable.

When invoking a process, the assignment of variables is specified as follows:

- If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification.
- Otherwise (the variables at the invoking and the process specification have the same name), the assignment is implied.

In the specification of a process, a specific coding block may be referred to by the variable name having a value equal to the address of the specific coding block.

6 Bitstream and picture formats, partitionings, scanning processes and neighbouring relationships

6.1 Bitstream formats

This subclause specifies the relationship between the network abstraction layer (NAL) unit stream and raw bitstream, either of which is referred to as the bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the raw bitstream file storage format. The NAL unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream.

The raw bitstream file storage format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a NAL unit length field to form a stream of bytes. Methods of framing the NAL units in a manner other than the use of the raw bitstream file storage

format are outside the scope of this document. The raw bitstream file storage format is specified in Annex B.

6.2 Source, decoded and output picture formats

This subclause specifies the relationship between source and decoded pictures that are given via the bitstream.

The video source that is represented by the bitstream is a sequence of pictures in decoding order.

The source and decoded pictures are each comprised of one or more sample arrays:

- Luma (Y) only (monochrome).
- Luma and two chroma (YCbCr or YCgCo).
- Green, blue, and red (GBR, also known as RGB).
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ).

For the convenience of notation and terminology in this document, the variables and terms associated with these arrays are referred to as luma (or L or Y) and chroma, where the two chroma arrays are referred to as Cb and Cr; regardless of the actual colour representation method in use. The actual colour representation method in use can be indicated in syntax that is specified in Annex E.

The variables SubWidthC and SubHeightC are specified in Table 2, depending on the chroma format sampling structure, which is specified through chroma_format_idc. Other values of chroma_format_idc, SubWidthC and SubHeightC may be specified in the future by ISO/IEC.

Table 2 — SubWidthC and SubHeightC values derived from chroma_format_idc

chroma_format_idc	Chroma format	SubWidthC	SubHeightC
0	Monochrome	1	1
1	4:2:0	2	2
2	4:2:2	2	1
3	4:4:4	1	1

In monochrome sampling, there is only one sample array, which is nominally considered the luma array.

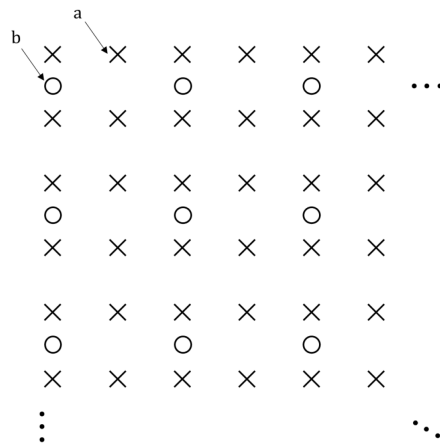
In 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array.

In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array.

In 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array.

The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 16, inclusive, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

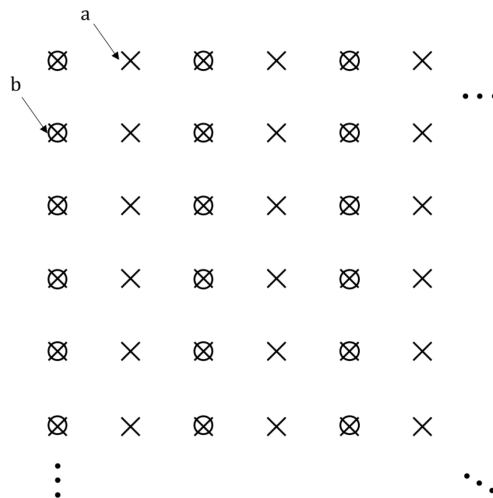
When the value of chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in pictures are shown in Figure 1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).



- a Location of luma sample.
b Location of chroma sample.

Figure 1 — Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture

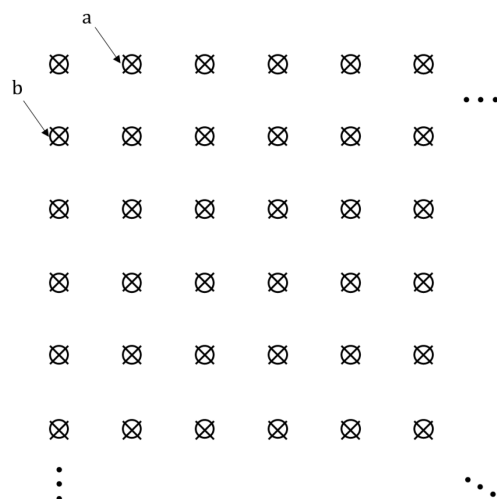
When the value of `chroma_format_idc` is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a picture are as shown in Figure 2.



- a Location of luma sample.
b Location of chroma sample.

Figure 2 — Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture

When the value of `chroma_format_idc` is equal to 3, all array samples are co-sited for all cases of pictures and the nominal locations in a picture are as shown in Figure 3.



- a Location of luma sample.
- b Location of chroma sample.

Figure 3 — Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

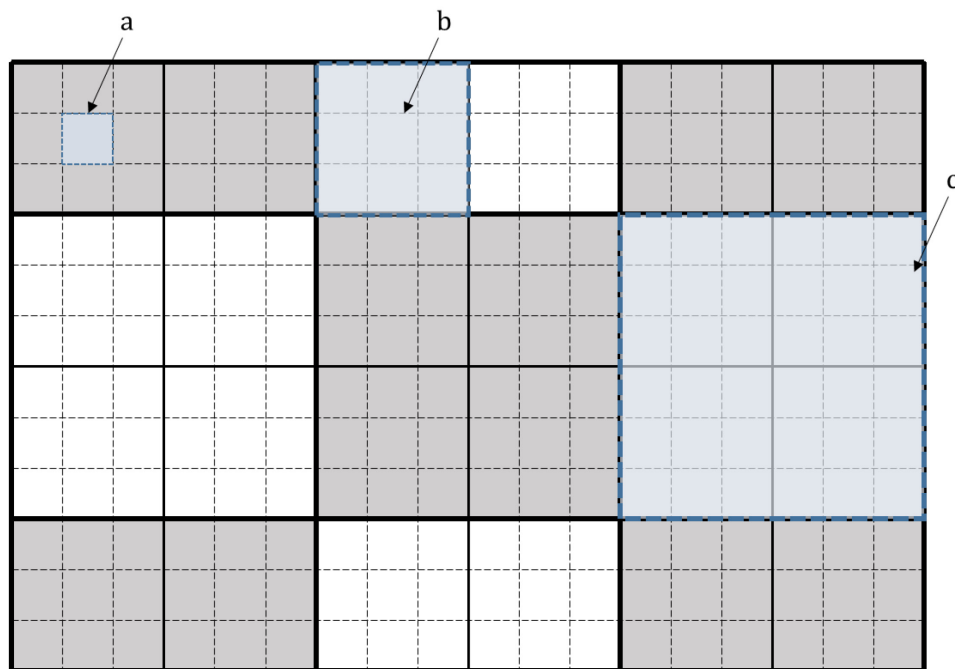
6.3 Partitioning of pictures, slices, tiles, and CTUs

6.3.1 Partitioning of pictures into slices and tiles

This subclause specifies how a picture is partitioned into slices and tiles.

Pictures are divided into slices and tiles. A tile is a group of CTUs that cover a rectangular region of a picture. A slice is a group of tiles that cover a rectangular region of a picture.

For example, a picture may be divided into 24 tiles (6 tile columns and 4 tile rows) and 9 slices as shown in Figure 4.

**Key**

- a CTU
- b tile
- c slice

Figure 4 — Picture with 18 by 12 luma CTUs that is partitioned into 24 tiles and 9 slices

6.3.2 Spatial or component-wise partitionings

The following divisions of processing elements form spatial or component-wise partitioning:

- the division of each picture into components;
- the division of each component into CTBs;
- the division of each picture into tile columns;
- the division of each picture into tile rows;
- the division of each tile column into tiles;
- the division of each tile row into tiles;
- the division of each tile into CTUs;
- the division of each picture into slices;
- the division of each slice into tiles;
- the division of each slice into CTUs; and
- the division of each CTU into CTBs.

6.4 Availability processes

6.4.1 Derivation process for neighbouring block availability

Input to this process is the luma location (x_{NbY} , y_{NbY}) covered by a neighbouring block relative to the top-left luma sample of the current picture.

Output of this process is the availability of the neighbouring block covering the location (x_{NbY} , y_{NbY}), denoted as $availableN$.

The neighbouring block availability $availableN$ is derived as follows:

- If one or more of the following conditions are true, $availableN$ is set equal to FALSE.
 - The neighbouring block is contained in a different tile than the current block.
 - x_{NbY} is less than 0.
 - y_{NbY} is less than 0.
 - x_{NbY} is greater than or equal to $pic_width_in_luma_samples$.
 - y_{NbY} is greater than or equal to $pic_height_in_luma_samples$.
 - $IsCoded[x_{NbY}][y_{NbY}]$ is equal to FALSE.
- Otherwise, $availableN$ is set equal to TRUE.

6.4.2 Derivation process for left and right neighbouring blocks availabilities

Inputs to this process are:

- the luma location (x_{Curr} , y_{Curr}) of the top-left sample of the current block relative to the top-left luma sample of the current picture, and
- a variable $nCbW$ specifying the width of the current block.

Output of this process is left and right availabilities of the neighbouring blocks, denoted as $availLR$.

A variable $availLR$, which indicates the left and right neighbouring blocks availabilities, is derived as follows:

- The left luma location (x_{NbL} , y_{NbL}) inside the neighbouring luma coding block is set equal to ($x_{Curr} - 1$, y_{Curr}).
- The right luma location (x_{NbR} , y_{NbR}) inside the neighbouring luma coding block is set equal to ($x_{Curr} + nCbW$, y_{Curr}).
- The derivation process for neighbouring block availability as specified in subclause 6.4.1 is invoked the left luma location (x_{NbL} , y_{NbL}) as input, and the output is assigned to the coding block availability flag, $availableL$.
- The derivation process for neighbouring block availability as specified in subclause 6.4.1 is invoked with the right luma location (x_{NbR} , y_{NbR}) as input, and the output is assigned to the coding block availability flag, $availableR$.

- The left and right neighbouring availability, availLR, is derived by:

$$\text{availLR} = \text{availableL} + \text{availableR} * 2 \quad (23)$$

availLR can be equal to LR_00, LR_10, LR_01, or LR_11, where LR_00 denotes the availability, availLR, equal to 0 when both left and right neighbouring blocks are not available; LR_10 denotes the availability, availLR, equal to 1 when left neighbouring block is available but right block is not available; LR_01 denotes the availability, availLR, equal to 2 when left neighbouring block is not available but the right block is available; LR_11 denotes the availability, availLR, equal to 3 when both left and right neighbouring block are available. When sps_suco_flag is equal to 0, availLR is always smaller than 2 (availLR is equal to LR_10 or availLR is equal to LR_00).

6.4.3 Derivation process for neighbouring block motion vector candidate availability

Input to this process is the luma location (xNbY, yNbY) covered by a neighbouring block relative to the top-left luma sample of the current picture.

Output of this process is the availability of the neighbouring block covering the location (xNbY, yNbY), denoted as availableN.

The neighbouring block availability availableN is derived as follows:

- If one or more of the following conditions are true, availableN is set equal to FALSE.
 - The neighbouring block is contained in a different tile than the current block.
 - xNbY is less than 0.
 - yNbY is less than 0.
 - xNbY is greater than or equal to pic_width_in_luma_samples.
 - yNbY is greater than or equal to pic_height_in_luma_samples.
 - IsCoded[xNbY][yNbY] is equal to FALSE.
 - The neighbouring block is coded in intra or intra block copy mode.
- Otherwise, availableN is set equal to TRUE.

6.4.4 Derivation process for ALF neighbouring block availability

Input to this process is the luma location (xNbY, yNbY) covered by a neighbouring block relative to the top-left luma sample of the current picture.

Output of this process is the availability of the neighbouring block covering the location (xNbY, yNbY), denoted as availableN.

The neighbouring block availability availableN is derived as follows:

- If one or more of the following conditions are true, availableN is set equal to FALSE.
 - xNbY is less than 0.
 - yNbY is less than 0.

- $xNbY$ is greater than or equal to $pic_width_in_luma_samples$.
- $yNbY$ is greater than or equal to $pic_height_in_luma_samples$.
- $IsCoded[xNbY][yNbY]$ is equal to FALSE.
- The neighbouring block is coded in intra or intra block copy mode.
- Otherwise, $availableN$ is set equal to TRUE.

6.5 Scanning processes

6.5.1 CTB raster and tile scanning process

The list $ColWidth[i]$ for i ranging from 0 to $num_tile_columns_minus1$, inclusive, specifying the width of the i -th tile column in units of CTBs, is derived as follows:

```

if( uniform_tile_spacing_flag )
    for( i = 0; i <= num_tile_columns_minus1; i++ )
        ColWidth[ i ] = ( ( i + 1 ) * PicWidthInCtbsY ) / ( num_tile_columns_minus1 + 1 ) -
            ( i * PicWidthInCtbsY ) / ( num_tile_columns_minus1 + 1 )
else {
    ColWidth[ num_tile_columns_minus1 ] = PicWidthInCtbsY
    for( i = 0; i < num_tile_columns_minus1; i++ ) {
        ColWidth[ i ] = tile_column_width_minus1[ i ] + 1
        ColWidth[ num_tile_columns_minus1 ] -= ColWidth[ i ]
    }
}
    
```

(24)

The list $RowHeight[j]$ for j ranging from 0 to $num_tile_rows_minus1$, inclusive, specifying the height of the j -th tile row in units of CTBs, is derived as follows:

```

if( uniform_tile_spacing_flag )
    for( j = 0; j <= num_tile_rows_minus1; j++ )
        RowHeight[ j ] = ( ( j + 1 ) * PicHeightInCtbsY ) / ( num_tile_rows_minus1 + 1 ) -
            ( j * PicHeightInCtbsY ) / ( num_tile_rows_minus1 + 1 )
else {
    RowHeight[ num_tile_rows_minus1 ] = PicHeightInCtbsY
    for( j = 0; j < num_tile_rows_minus1; j++ ) {
        RowHeight[ j ] = tile_row_height_minus1[ j ] + 1
        RowHeight[ num_tile_rows_minus1 ] -= RowHeight[ j ]
    }
}
    
```

(25)

The list $ColBd[i]$ for i ranging from 0 to $num_tile_columns_minus1 + 1$, inclusive, specifying the location of the i -th tile column boundary in units of CTBs, is derived as follows:

```

for( ColBd[ 0 ] = 0, i = 0; i <= num_tile_columns_minus1; i++ )
    ColBd[ i + 1 ] = ColBd[ i ] + ColWidth[ i ]
    
```

(26)

The list $RowBd[j]$ for j ranging from 0 to $num_tile_rows_minus1 + 1$, inclusive, specifying the location of the j -th tile row boundary in units of CTBs, is derived as follows:

```

for( RowBd[ 0 ] = 0, j = 0; j <= num_tile_rows_minus1; j++ )
    RowBd[ j + 1 ] = RowBd[ j ] + RowHeight[ j ]

```

(27)

The list CtbAddrRsToTs[ctbAddrRs] for ctbAddrRs ranging from 0 to PicSizeInCtbsY – 1, inclusive, specifying the conversion from a CTB address in CTB raster scan of a picture to a CTB address in tile scan, is derived as follows:

```

for( ctbAddrRs = 0; ctbAddrRs < PicSizeInCtbsY; ctbAddrRs++ ) {
    tbX = ctbAddrRs % PicWidthInCtbsY
    tbY = ctbAddrRs / PicWidthInCtbsY
    for( i = 0; i <= num_tile_columns_minus1; i++ )
        if( tbX >= ColBd[ i ] )
            tileX = i
    for( j = 0; j <= num_tile_rows_minus1; j++ )
        if( tbY >= RowBd[ j ] )
            tileY = j
    CtbAddrRsToTs[ ctbAddrRs ] = 0
    for( i = 0; i < tileX; i++ )
        CtbAddrRsToTs[ ctbAddrRs ] += RowHeight[ tileY ] * ColWidth[ i ]
    for( j = 0; j < tileY; j++ )
        CtbAddrRsToTs[ ctbAddrRs ] += PicWidthInCtbsY * RowHeight[ j ]
    CtbAddrRsToTs[ ctbAddrRs ] +=
    ( tbY – RowBd[ tileY ] ) * ColWidth[ tileX ] + tbX – ColBd[ tileX ]
}

```

(28)

The list CtbAddrTsToRs[ctbAddrTs] for ctbAddrTs ranging from 0 to PicSizeInCtbsY – 1, inclusive, specifying the conversion from a CTB address in tile scan to a CTB address in CTB raster scan of a picture, is derived as follows:

```

for( ctbAddrRs = 0; ctbAddrRs < PicSizeInCtbsY; ctbAddrRs++ )
    CtbAddrTsToRs[ CtbAddrRsToTs[ ctbAddrRs ] ] = ctbAddrRs

```

(29)

The list TileId[ctbAddrTs] for ctbAddrTs ranging from 0 to PicSizeInCtbsY – 1, inclusive, specifying the conversion from a CTB address in tile scan to a tile ID, is derived as follows:

```

for( j = 0, tileIdx = 0; j <= num_tile_rows_minus1; j++ )
    for( i = 0; i <= num_tile_columns_minus1; i++, tileIdx++ )
        for( y = RowBd[ j ]; y < RowBd[ j + 1 ]; y++ )
            for( x = ColBd[ i ]; x < ColBd[ i + 1 ]; x++ )
                TileId[ CtbAddrRsToTs[ y * PicWidthInCtbsY + x ] ] =
                explicit_tile_id_flag ? tile_id_val[ i ][ j ] : tileIdx

```

(30)

The list NumCtusInTile[tileIdx] for tileIdx ranging from 0 to PicSizeInCtbsY – 1, inclusive, specifying the conversion from a tile index to the number of CTUs in the tile, is derived as follows:

```

for( j = 0, tileIdx = 0; j <= num_tile_rows_minus1; j++ )
    for( i = 0; i <= num_tile_columns_minus1; i++, tileIdx++ )
        NumCtusInTile[ tileIdx ] = ColWidth[ i ] * RowHeight[ j ]

```

(31)

The set TileIdToIdx[tileId] for a set of NumTilesInPic tileId values specifying the conversion from a tile ID to a tile index and the list FirstCtbAddrTs[tileIdx] for tileIdx ranging from 0 to NumTilesInPic – 1, inclusive, specifying the conversion from a tile ID to the CTB address in tile scan of the first CTB in the tile are derived as follows:

```

for( ctbAddrTs = 0, tileIdx = 0, tileStartFlag = 1; ctbAddrTs < PicSizeInCtbsY; ctbAddrTs++ ) {
    if( tileStartFlag ) {
        TileIdToIdx[ TileId[ ctbAddrTs ] ] = tileIdx
        FirstCtbAddrTs[ tileIdx ] = ctbAddrTs
        tileStartFlag = 0
    }
    tileEndFlag = ctbAddrTs == PicSizeInCtbsY - 1 || TileId[ ctbAddrTs + 1 ] !=
    TileId[ ctbAddrTs ]
    if( tileEndFlag ) {
        tileIdx++
        tileStartFlag = 1
    }
}

```

(32)

The values of `ColumnWidthInLumaSamples[ i ]`, specifying the width of the *i*-th tile column in units of luma samples, are set equal to `ColWidth[ i ] << CtbLog2SizeY` for *i* ranging from 0 to `num_tile_columns_minus1`, inclusive.

The values of `RowHeightInLumaSamples[ j ]`, specifying the height of the *j*-th tile row in units of luma samples, are set equal to `RowHeight[ j ] << CtbLog2SizeY` for *j* ranging from 0 to `num_tile_rows_minus1`, inclusive.

6.5.2 Zig-zag scan order 1D array initialization process

Inputs to this process are:

- a variable `blkWidth` specifying the width of block, and
- a variable `blkHeight` specifying the height of block.

Output of this process is the array `zigZagScan[ sPos ]`.

The array index `sPos` specifies the scan position ranging from 0 to $(\text{blkWidth} * \text{blkHeight}) - 1$. Depending on the value of `blkWidth` and `blkHeight`, the array `zigZagScan` is derived as follows:

```

pos = 0
zigZagScan[ pos ] = 0
pos++
for( line = 1; line < ( blkWidth + blkHeight - 1 ); line++ ) {
    if( line % 2 ) {
        x = Min( line, blkWidth - 1 )
        y = Max( 0, line - ( blkWidth - 1 ) )
        while( x >= 0 && y < blkHeight ) {
            zigZagScan[ pos ] = y * blkWidth + x
            pos++
            x--
            y++
        }
    }
    else{
        y = Min( line, -1 )
        x = Max( 0, line - ( blkHeight - 1 ) )
        while( y >= 0 && x < blkWidth ) {
            zigZagScan[ pos ] = y * blkWidth + x

```

(33)

```

        pos++
        x++
        y--
    }
}

```

6.5.3 Inverse scan order 1D array initialization process

Inputs to this process are:

- a variable blkWidth specifying the width of block, and
- a variable blkHeight specifying the height of block.

Output of this process is the array inverseScan[rPos].

The array index rPos specifies the raster scan position ranging from 0 to (blkWidth * blkHeight) – 1. Depending on the value of blkWidth and blkHeight, the array inverseScan is derived as follows:

- The variable forwardScan is derived by invoking zig-zag scan order 1D array initialization process as specified in subclause 6.5.2 with input parameters blkWidth and blkHeight.
- The output variable inverseScan is derived as follows:

```

for( pos = 0; pos < blkWidth * blkHeight; pos++ ) {
    inverseScan[ forwardScan[ pos ] ] = pos
}

```

(34)

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

The syntax tables specify a superset of the syntax of all allowed bitstreams.

NOTE An actual decoder must implement some means for identifying entry points into the bitstream and some means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified in this document.

Table 3 lists examples of the syntax specification format. When **syntax_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

Table 3 — Examples of the syntax specification format

	Descriptor
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples */	
syntax_element	ue(v)
conditioning statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while(condition)	
statement	
/* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do	
statement	
while(condition)	
/* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	
if(condition)	
primary statement	
else	
alternative statement	
/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

7.2 Specification of syntax functions and descriptors

The functions presented in this document are used in the syntactical description. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

`byte_aligned( )` is specified as follows:

- If the current position in the bitstream is on a byte boundary, i.e., the next bit in the bitstream is the first bit in a byte, the return value of `byte_aligned( )` is equal to TRUE.
- Otherwise, the return value of `byte_aligned( )` is equal to FALSE.

`more_data_in_payload( )` is specified as follows:

- If `byte_aligned( )` is equal to TRUE and the current position in the `sei_payload( )` syntax structure is $8 * \text{payloadSize}$ bits from the beginning of the `sei_payload( )` syntax structure, the return value of `more_data_in_payload( )` is equal to FALSE.
- Otherwise, the return value of `more_data_in_payload( )` is equal to TRUE.

`more_rbsp_data( )` is specified as follows:

- If there is no more data in the raw byte sequence payload (Rbsp), the return value of `more_rbsp_data( )` is equal to FALSE.
- Otherwise, the Rbsp data are searched for the last (least significant, right-most) bit equal to 1 that is present in the Rbsp. Given the position of this bit, which is the first bit (`rbp_stop_one_bit`) of the `rbp_trailing_bits( )` syntax structure, the following applies:
 - If there is more data in an Rbsp before the `rbp_trailing_bits( )` syntax structure, the return value of `more_rbsp_data( )` is equal to TRUE.
 - Otherwise, the return value of `more_rbsp_data( )` is equal to FALSE.

The method for enabling determination of whether there is more data in the Rbsp is specified by the application (or in Annex B for applications that use the raw bitstream file storage format).

`more_rbsp_trailing_data( )` is specified as follows:

- If there is more data in an Rbsp, the return value of `more_rbsp_trailing_data( )` is equal to TRUE.
- Otherwise, the return value of `more_rbsp_trailing_data( )` is equal to FALSE.

`next_bits( n )` provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next n bits in the bitstream with n being its argument.

`payload_extension_present( )` is specified as follows:

- If the current position in the `sei_payload( )` syntax structure is not the position of the last (least significant, right-most) bit that is equal to 1 that is less than $8 * \text{payloadSize}$ bits from the beginning of the syntax structure (i.e., the position of the `payload_bit_equal_to_one` syntax element), the return value of `payload_extension_present( )` is equal to TRUE.
- Otherwise, the return value of `payload_extension_present( )` is equal to FALSE.

`read_bits( n )` reads the next `n` bits from the bitstream and advances the bitstream pointer by `n` bit positions. When `n` is equal to 0, `read_bits( n )` is specified to return a value equal to 0 and to not advance the bitstream pointer.

The following descriptors specify the parsing process of each syntax element:

- `ae(v)`: context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in subclause 9.3.
- `b(8)`: byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function `read_bits( 8 )`.
- `f(n)`: fixed-pattern bit string using `n` bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )`.
- `i(n)`: signed integer using `n` bits. When `n` is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )` interpreted as a two's complement integer representation with most significant bit written first.
- `se(v)`: signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2
- `u(n)`: unsigned integer using `n` bits. When `n` is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )` interpreted as a binary representation of an unsigned integer with most significant bit written first.
- `ue(v)`: unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.
- `uek(v)`: unsigned integer `k`-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2 with the order `k` defined in the semantics of the syntax element.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

7.3.1.1 General NAL unit syntax

<code>nal_unit(NumBytesInNalUnit) {</code>	Descriptor
<code>nal_unit_header()</code>	
<code>NumBytesInRbsp = 0</code>	
<code>for(i = 2; i < NumBytesInNalUnit; i++)</code>	
<code> <code>rbsp_byte[NumBytesInRbsp++]</code></code>	<code>b(8)</code>
<code>}</code>	

7.3.1.2 NAL unit header syntax

<code>nal_unit_header() {</code>	Descriptor
<code>forbidden_zero_bit</code>	<code>f(1)</code>
<code>nal_unit_type_plus1</code>	<code>u(6)</code>
<code>nuh_temporal_id</code>	<code>u(3)</code>
<code>nuh_reserved_zero_5bits</code>	<code>u(5)</code>
<code>nuh_extension_flag</code>	<code>u(1)</code>
<code>}</code>	

7.3.2 Raw byte sequence payloads, trailing bits and byte alignment syntax

7.3.2.1 SPS RBSP syntax

<code>seq_parameter_set_rbsp() {</code>	Descriptor
<code>sps_seq_parameter_set_id</code>	<code>ue(v)</code>
<code>profile_idc</code>	<code>u(8)</code>
<code>level_idc</code>	<code>u(8)</code>
<code>toolset_idc_h</code>	<code>u(32)</code>
<code>toolset_idc_l</code>	<code>u(32)</code>
<code>chroma_format_idc</code>	<code>ue(v)</code>
<code>pic_width_in_luma_samples</code>	<code>ue(v)</code>
<code>pic_height_in_luma_samples</code>	<code>ue(v)</code>
<code>bit_depth_luma_minus8</code>	<code>ue(v)</code>
<code>bit_depth_chroma_minus8</code>	<code>ue(v)</code>
<code>sps_btt_flag</code>	<code>u(1)</code>
<code>if(sps_btt_flag) {</code>	
<code>log2_ctu_size_minus5</code>	<code>ue(v)</code>
<code>log2_min_cb_size_minus2</code>	<code>ue(v)</code>
<code>log2_diff_ctu_max_14_cb_size</code>	<code>ue(v)</code>
<code>log2_diff_ctu_max_tt_cb_size</code>	<code>ue(v)</code>
<code>log2_diff_min_cb_min_tt_cb_size_minus2</code>	<code>ue(v)</code>
<code>}</code>	
<code>sps_suco_flag</code>	<code>u(1)</code>
<code>if(sps_suco_flag) {</code>	
<code>log2_diff_ctu_size_max_suco_cb_size</code>	<code>ue(v)</code>
<code>log2_diff_max_suco_min_suco_cb_size</code>	<code>ue(v)</code>
<code>}</code>	
<code>sps_admvp_flag</code>	<code>u(1)</code>
<code>if(sps_admvp_flag) {</code>	
<code>sps_affine_flag</code>	<code>u(1)</code>
<code>sps_amvr_flag</code>	<code>u(1)</code>
<code>sps_dmvr_flag</code>	<code>u(1)</code>
<code>sps_mmvd_flag</code>	<code>u(1)</code>

sps_hmvp_flag	u(1)
}	
sps_eipd_flag	u(1)
if(sps_eipd_flag) {	
sps_ibc_flag	u(1)
if(sps_ibc_flag)	
log2_max_ibc_cand_size_minus2	ue(v)
}	
sps_cm_init_flag	u(1)
if(sps_cm_init_flag)	
sps_adcc_flag	u(1)
sps_iqt_flag	u(1)
if(sps_iqt_flag)	
sps_ats_flag	u(1)
sps_addb_flag	u(1)
sps_alf_flag	u(1)
sps_htdf_flag	u(1)
sps_rpl_flag	u(1)
sps_pocs_flag	u(1)
sps_dquant_flag	u(1)
sps_dra_flag	u(1)
if(sps_pocs_flag)	
log2_max_pic_order_cnt_lsb_minus4	ue(v)
if(!sps_pocs_flag !sps_rpl_flag) {	
log2_sub_gop_length	ue(v)
if(log2_sub_gop_length == 0)	
log2_ref_pic_gap_length	ue(v)
}	
if(!sps_rpl_flag)	
max_num_tid0_ref_pics	ue(v)
else {	
sps_max_dec_pic_buffering_minus1	ue(v)
long_term_ref_pics_flag	u(1)
rpl1_same_as_rpl0_flag	u(1)
for(i = 0; i < !rpl1_same_as_rpl0_flag ? 2 : 1; i++) {	
num_ref_pic_lists_in_sps[i]	ue(v)
for(j = 0; j < num_ref_pic_lists_in_sps[i]; j++)	
ref_pic_list_struct(i, j, long_term_ref_pics_flag)	
}	
}	
picture_cropping_flag	u(1)
if(picture_cropping_flag) {	
picture_crop_left_offset	ue(v)
picture_crop_right_offset	ue(v)
picture_crop_top_offset	ue(v)
picture_crop_bottom_offset	ue(v)

}	
if(ChromaArrayType != 0) {	
chroma_qp_table_present_flag	u(1)
if(chroma_qp_table_present_flag)	
same_qp_table_for_chroma	u(1)
global_offset_flag	u(1)
for(i = 0; i < same_qp_table_for_chroma ? 1 : 2; i++) {	
num_points_in_qp_table_minus1[i]	ue(v)
for(j = 0; j <= num_points_in_qp_table_minus1[i]; j++)	
delta_qp_in_val_minus1[i][j]	u(6)
delta_qp_out_val[i][j]	se(v)
}	
}	
}	
vui_parameters_present_flag	u(1)
if(vui_parameters_present_flag)	
vui_parameters()	
rbsp_trailing_bits()	
}	

7.3.2.2 PPS RBSP syntax

	Descriptor
pic_parameter_set_rbsp() {	
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
for(i = 0; i < 2; i++)	
num_ref_idx_default_active_minus1[i]	ue(v)
additional_lt_poc_lsb_len	ue(v)
rpl1_idx_present_flag	u(1)
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
uniform_tile_spacing_flag	u(1)
if(!uniform_tile_spacing_flag) {	
for(i = 0; i < num_tile_columns_minus1; i++)	
tile_column_width_minus1[i]	ue(v)
for(i = 0; i < num_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)
}	
loop_filter_across_tiles_enabled_flag	u(1)
tile_offset_len_minus1	ue(v)
}	
tile_id_len_minus1	ue(v)
explicit_tile_id_flag	u(1)
if(explicit_tile_id_flag)	
for(i = 0; i <= num_tile_rows_minus1; i++)	
for(j = 0; j <= num_tile_columns_minus1; j++)	
tile_id_val[i][j]	u(v)
pic_dra_enabled_flag	u(1)
if(pic_dra_enabled_flag)	
pic_dra_aps_id	u(5)
arbitrary_slice_present_flag	u(1)
constrained_intra_pred_flag	u(1)
cu_qp_delta_enabled_flag	u(1)
if(cu_qp_delta_enabled_flag)	
log2_cu_qp_delta_area_minus6	ue(v)
rbsp_trailing_bits()	
}	

7.3.2.3 APS RBSP syntax

adaptation_parameter_set_rbsp() {	Descriptor
adaptation_parameter_set_id	u(5)
aps_params_type	u(3)
if(aps_params_type == ALF_APS)	
alf_data()	
else if(aps_params_type == DRA_APS)	
dra_data()	
aps_extension_flag	u(1)
if(aps_extension_flag)	
while(more_rbsp_data())	
aps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

7.3.2.4 Filler data RBSP syntax

filler_data_rbsp() {	Descriptor
while(next_bits(8) == 0xFF)	
ff_byte // equal to 0xFF	f(8)
rbsp_trailing_bits()	
}	

7.3.2.5 Supplemental enhancement information RBSP syntax

sei_rbsp() {	Descriptor
do	
sei_message()	
while(more_rbsp_data())	
rbsp_trailing_bits()	
}	

7.3.2.6 Slice layer RBSP syntax

slice_layer_rbsp() {	Descriptor
slice_header()	
slice_data()	
rbsp_slice_trailing_bits()	
}	

7.3.2.7 RBSP slice trailing bits syntax

rbsp_slice_trailing_bits() {	Descriptor
rbsp_trailing_bits()	
while(more_rbsp_trailing_data())	
cabac_zero_word /* equal to 0x0000 */	f(16)
}	

7.3.2.8 RBSP trailing bits syntax

rbsp_trailing_bits() {	Descriptor
rbsp_stop_one_bit /* equal to 1 */	f(1)
while(!byte_aligned())	
rbsp_alignment_zero_bit /* equal to 0 */	f(1)
}	

7.3.2.9 Byte alignment syntax

byte_alignment() {	Descriptor
alignment_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned())	
alignment_bit_equal_to_zero /* equal to 0 */	f(1)
}	

7.3.3 Supplemental enhancement information message syntax

sei_message() {	Descriptor
payloadType = 0	
do {	
payload_type_byte	u(8)
payloadType += payload_type_byte	
} while(payload_type_byte == 0xFF)	
payloadSize = 0	
do {	
payload_size_byte	u(8)
payloadSize += payload_size_byte	
} while(payload_size_byte == 0xFF)	
sei_payload(payloadType, payloadSize)	
}	

7.3.4 Slice header syntax

slice_header() {	Descriptor
slice_pic_parameter_set_id	ue(v)
if(!single_tile_in_pic_flag) {	
single_tile_in_slice_flag	u(1)
first_tile_id	u(v)
}	
if(!single_tile_in_slice_flag) {	
if(arbitrary_slice_present_flag)	
arbitrary_slice_flag	u(1)
if(!arbitrary_slice_flag)	
last_tile_id	u(v)
else {	
num_remaining_tiles_in_slice_minus1	ue(v)
for(i = 0; i < NumTilesInSlice - 1; i++)	
delta_tile_id_minus1[i]	ue(v)
}	
}	
slice_type	ue(v)
if(nal_unit_type == IDR_NUT)	
no_output_of_prior_pics_flag	u(1)
if(sps_mmvd_flag && (slice_type == B slice_type == P))	
mmvd_group_enable_flag	u(1)
if(sps_alf_flag) {	
slice_alf_enabled_flag	u(1)
if(slice_alf_enabled_flag) {	
slice_alf_luma_aps_id	u(5)
slice_alf_map_flag	u(1)
slice_alf_chroma_idc	u(2)
if((ChromaArrayType == 1 ChromaArrayType == 2) && slice_alf_chroma_idc > 0)	
slice_alf_chroma_aps_id	u(5)
}	
if(ChromaArrayType == 3) {	
if(!slice_alf_enabled_flag)	
slice_alf_chroma_idc	u(2)
if(sliceChromaAlfEnabledFlag) {	
slice_alf_chroma_aps_id	u(5)
slice_alf_chroma_map_flag	u(1)
}	
if(sliceChroma2AlfEnabledFlag) {	
slice_alf_chroma2_aps_id	u(5)
slice_alf_chroma2_map_flag	u(1)
}	
}	

if(NalUnitType != IDR_NUT) {	
if(sps_pocs_flag)	
slice_pic_order_cnt_lsb	u(v)
if(sps_rpl_flag) {	
for(i = 0; i < 2; i++) {	
if(num_ref_pic_lists_in_sps[i] > 0 && (i == 0 (i == 1 && rpl1_idx_present_flag)))	
ref_pic_list_sps_flag[i]	u(1)
if(ref_pic_list_sps_flag[i]) {	
if(num_ref_pic_lists_in_sps[i] > 1 && (i == 0 (i == 1 && rpl1_idx_present_flag)))	
ref_pic_list_idx[i]	u(v)
} else	
ref_pic_list_struct(i, num_ref_pic_lists_in_sps[i], long_term_ref_pics_flag)	
for(j = 0; j < num_ltrp_entries[i][SliceRplsIdx[i]]; j++) {	
additional_poc_lsb_present_flag[i][j]	u(1)
if(additional_poc_lsb_present_flag[i][j])	
additional_poc_lsb_val[i][j]	u(v)
}	
}	
}	
if(slice_type == P slice_type == B) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag)	
for(i = 0; i < (slice_type == B ? 2 : 1); i++)	
num_ref_idx_active_minus1[i]	ue(v)
if(sps_admvp_flag) {	
temporal_mvp_assigned_flag	u(1)
if(temporal_mvp_assigned_flag) {	
if(slice_type == B) {	
col_pic_list_idx	u(1)
col_source_mvp_list_idx	u(1)
}	
col_pic_ref_idx	ue(v)
}	
}	
}	
}	
slice_deblocking_filter_flag	u(1)
if(slice_deblocking_filter_flag && sps_addb_flag) {	
slice_alpha_offset	se(v)
slice_beta_offset	se(v)
}	
slice_qp	u(6)
slice_cb_qp_offset	se(v)
slice_cr_qp_offset	se(v)

if(!single_tile_in_slice_flag)	
for(i = 0; i < NumTilesInSlice - 1; i++)	
entry_point_offset_minus1[i]	u(v)
byte_alignment()	
}	

7.3.5 Adaptive loop filter data syntax

alf_data() {	Descriptor
alf_luma_filter_signal_flag	u(1)
alf_chroma_filter_signal_flag	u(1)
if(alf_luma_filter_signal_flag) {	
alf_luma_num_filters_signalled_minus1	ue(v)
alf_luma_type_flag	u(1)
if(alf_luma_num_filters_signalled_minus1 > 0) {	
for(i = 0; i < NumAlfFilters; i++)	
alf_luma_coeff_delta_idx[i]	u(v)
}	
alf_luma_fixed_filter_usage_pattern	uek(v)
if(alf_luma_fixed_filter_usage_pattern == 2) {	
for(i = 0; i < NumAlfFilters; i++)	
alf_luma_fixed_filter_usage_flag[i]	u(1)
}	
if(alf_luma_fixed_filter_usage_pattern > 0) {	
for(i = 0; i < NumAlfFilters; i++) {	
if(alf_luma_fixed_filter_usage_flag[i])	
alf_luma_fixed_filter_set_idx[i]	u(4)
}	
}	
alf_luma_coeff_delta_flag	u(1)
if(!alf_luma_coeff_delta_flag && alf_luma_num_filters_signalled_minus1 > 0)	
alf_luma_coeff_delta_prediction_flag	u(1)
alf_luma_min_eg_order_minus1	ue(v)
for(i = 0; i < LumaMaxGolombIdx; i++)	
alf_luma_eg_order_increase_flag[i]	u(1)
if(alf_luma_coeff_delta_flag) {	
for(i = 0; i < NumSignalledFilter; i++)	
alf_luma_coeff_flag[i]	u(1)
}	

for(i = 0; i < NumSignalledFilter; i++) {	
if(alf_luma_coeff_flag[i]) {	
for(j = 0; j < NumAlfCoefs - 1; j++) {	
alf_luma_coeff_delta_abs[i][j]	uek(v)
if(alf_luma_coeff_delta_abs[i][j])	
alf_luma_coeff_delta_sign_flag[i][j]	u(1)
}	
}	
}	
}	
if(alf_chroma_filter_signal_flag) {	
alf_chroma_min_eg_order_minus1	ue(v)
for(i = 0; i < ChromaMaxGolombIdx; i++)	
alf_chroma_eg_order_increase_flag[i]	u(1)
for(j = 0; j < 6; j++) {	
alf_chroma_coeff_abs[j]	uek(v)
if(alf_chroma_coeff_abs[j] > 0)	
alf_chroma_coeff_sign_flag[j]	u(1)
}	
}	
}	
}	

7.3.6 DRA data syntax

dra_data() {	Descriptor
dra_descriptor1	u(4)
dra_descriptor2	u(4)
dra_number_ranges_minus1	ue(v)
dra_equal_ranges_flag	u(1)
dra_global_offset	u(10)
if(dra_equal_ranges_flag)	
dra_delta_range[0]	u(10)
else	
for(j = 0; j <= dra_number_ranges_minus1; j++)	
dra_delta_range[j]	u(10)
for(j = 0; j <= dra_number_ranges_minus1; j++)	
dra_scale_value[j]	u(v)
dra_cb_scale_value	u(v)
dra_cr_scale_value	u(v)
dra_table_idx	ue(v)
}	