

International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.264

(11/2007)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS
Infrastructure of audiovisual services – Coding of moving
video

**Advanced video coding for generic audiovisual
services**

ITU-T Recommendation H.264

ITU-T



ITU-T H-SERIES RECOMMENDATIONS
AUDIOVISUAL AND MULTIMEDIA SYSTEMS

CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100–H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	
General	H.200–H.219
Transmission multiplexing and synchronization	H.220–H.229
Systems aspects	H.230–H.239
Communication procedures	H.240–H.259
Coding of moving video	H.260–H.279
Related systems aspects	H.280–H.299
Systems and terminal equipment for audiovisual services	H.300–H.349
Directory services architecture for audiovisual and multimedia services	H.350–H.359
Quality of service architecture for audiovisual and multimedia services	H.360–H.369
Supplementary services for multimedia	H.450–H.499
MOBILITY AND COLLABORATION PROCEDURES	
Overview of Mobility and Collaboration, definitions, protocols and procedures	H.500–H.509
Mobility for H-Series multimedia systems and services	H.510–H.519
Mobile multimedia collaboration applications and services	H.520–H.529
Security for mobile multimedia systems and services	H.530–H.539
Security for mobile multimedia collaboration applications and services	H.540–H.549
Mobility interworking procedures	H.550–H.559
Mobile multimedia collaboration inter-working procedures	H.560–H.569
BROADBAND AND TRIPLE-PLAY MULTIMEDIA SERVICES	
Broadband multimedia services over VDSL	H.610–H.619

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation H.264

Advanced video coding for generic audiovisual services

Summary

This Recommendation | International Standard represents an evolution of the existing video coding standards (H.261, H.262, and H.263) and it was developed in response to the growing need for higher compression of moving pictures for various applications such as videoconferencing, digital storage media, television broadcasting, Internet streaming, and communication. It is also designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments. The use of this Recommendation | International Standard allows motion video to be manipulated as a form of computer data and to be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

The revision approved 2005-03 contained modifications of the video coding standard to add four new profiles, referred to as the High, High 10, High 4:2:2, and High 4:4:4 profiles, to improve video quality capability and to extend the range of applications addressed by the standard (for example, by including support for a greater range of picture sample precision and higher-resolution chroma formats). Additionally, a definition of new types of supplemental data was specified to further broaden the applicability of the video coding standard. Finally, a number of corrections to errors in the published text were included.

Corrigendum 1 to ITU-T Rec. H.264 corrected and updated various minor aspects to bring the ITU-T version of the text up to date relative to the April 2005 output status approved as a new edition of the corresponding jointly-developed and technically-aligned text ISO/IEC 14496-10. It additionally fixed a number of minor errors and needs for clarification and defined three previously-reserved sample aspect ratio indicators.

Amendment 1 "Support of additional colour spaces and removal of the High 4:4:4 Profile" contained alterations to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify the support of additional colour spaces and to remove the definition of the High 4:4:4 Profile.

NOTE – ITU-T Rec. H.264 is a twin text with ISO/IEC 14496-10 and this amendment is published in two different documents in the ISO/IEC series:

- The removal of the High 4:4:4 profile is found in ISO/IEC 14496-10:2005/Cor.2.
- The specification for support of additional colour space will be found in ISO/IEC 14496-10:2005/Amd.1.

Amendment 2 "New profiles for professional applications" contained extensions to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify the support of five additional profiles intended primarily for professional applications (the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles) and two new types of supplemental enhancement information (SEI) messages (the post-filter hint SEI message and the tone mapping information SEI message).

Amendment 3 "Scalable video coding" contained extensions to ITU-T Rec. H.264 | ISO/IEC 14496-10 Advanced Video Coding to specify a scalable video coding extension in three profiles (the Scalable Baseline, Scalable High, and Scalable High Intra profiles).

The H.264 edition published in 2005-11 included the text approved 2005-03 and its Corrigendum 1 approved 2005-09. H.264 (2005) Amd.2 (2007) was available only as pre-published text since it was superseded by H.264 Amd.3 (11/2007) before its publication; further, H.264 Amd.3 was not published separately. This third edition integrates into the H.264 edition published in 2005-11 all changes approved in Amendments 1 (2006-06), 2 (2007-04) and 3 (2007-11).

Source

ITU-T Recommendation H.264 was approved on 22 November 2007 by ITU-T Study Group 16 (2005-2008) under the ITU-T Recommendation A.8 procedure.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2008

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

CONTENTS

	Page
Foreword	xix
0 Introduction	1
0.1 Prologue	1
0.2 Purpose	1
0.3 Applications	1
0.4 Publication and versions of this specification	1
0.5 Profiles and levels	2
0.6 Overview of the design characteristics	3
0.6.1 Predictive coding	3
0.6.2 Coding of progressive and interlaced video	3
0.6.3 Picture partitioning into macroblocks and smaller partitions	3
0.6.4 Spatial redundancy reduction	4
0.7 How to read this specification	4
1 Scope	4
2 Normative references	4
3 Definitions	5
4 Abbreviations	13
5 Conventions	13
5.1 Arithmetic operators	13
5.2 Logical operators	14
5.3 Relational operators	14
5.4 Bit-wise operators	14
5.5 Assignment operators	15
5.6 Range notation	15
5.7 Mathematical functions	15
5.8 Variables, syntax elements, and tables	16
5.9 Text description of logical operations	17
5.10 Processes	18
6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships	18
6.1 Bitstream formats	18
6.2 Source, decoded, and output picture formats	18
6.3 Spatial subdivision of pictures and slices	23
6.4 Inverse scanning processes and derivation processes for neighbours	24
6.4.1 Inverse macroblock scanning process	24
6.4.2 Inverse macroblock partition and sub-macroblock partition scanning process	25
6.4.2.1 Inverse macroblock partition scanning process	26
6.4.2.2 Inverse sub-macroblock partition scanning process	26
6.4.3 Inverse 4x4 luma block scanning process	27
6.4.4 Inverse 4x4 Cb or Cr block scanning process for ChromaArrayType equal to 3	27
6.4.5 Inverse 8x8 luma block scanning process	27
6.4.6 Inverse 8x8 Cb or Cr block scanning process for ChromaArrayType equal to 3	28
6.4.7 Derivation process of the availability for macroblock addresses	28
6.4.8 Derivation process for neighbouring macroblock addresses and their availability	28
6.4.9 Derivation process for neighbouring macroblock addresses and their availability in MBAFF frames	29
6.4.10 Derivation processes for neighbouring macroblocks, blocks, and partitions	29
6.4.10.1 Derivation process for neighbouring macroblocks	30
6.4.10.2 Derivation process for neighbouring 8x8 luma block	30
6.4.10.3 Derivation process for neighbouring 8x8 chroma blocks for ChromaArrayType equal to 3	31
6.4.10.4 Derivation process for neighbouring 4x4 luma blocks	31
6.4.10.5 Derivation process for neighbouring 4x4 chroma blocks	32
6.4.10.6 Derivation process for neighbouring 4x4 chroma blocks for ChromaArrayType equal to 3	32
6.4.10.7 Derivation process for neighbouring partitions	32
6.4.11 Derivation process for neighbouring locations	34
6.4.11.1 Specification for neighbouring locations in fields and non-MBAFF frames	34
6.4.11.2 Specification for neighbouring locations in MBAFF frames	35
7 Syntax and semantics	37
7.1 Method of specifying syntax in tabular form	37
7.2 Specification of syntax functions, categories, and descriptors	38

7.3	<i>Syntax in tabular form</i>	39
7.3.1	NAL unit syntax	39
7.3.2	Raw byte sequence payloads and RBSP trailing bits syntax	39
7.3.2.1	Sequence parameter set RBSP syntax	39
7.3.2.1.1	Sequence parameter set data syntax	40
7.3.2.1.1.1	Scaling list syntax	41
7.3.2.1.2	Sequence parameter set extension RBSP syntax	41
7.3.2.2	Picture parameter set RBSP syntax	42
7.3.2.3	Supplemental enhancement information RBSP syntax	43
7.3.2.3.1	Supplemental enhancement information message syntax	43
7.3.2.4	Access unit delimiter RBSP syntax	43
7.3.2.5	End of sequence RBSP syntax	44
7.3.2.6	End of stream RBSP syntax	44
7.3.2.7	Filler data RBSP syntax	44
7.3.2.8	Slice layer without partitioning RBSP syntax	44
7.3.2.9	Slice data partition RBSP syntax	44
7.3.2.9.1	Slice data partition A RBSP syntax	44
7.3.2.9.2	Slice data partition B RBSP syntax	45
7.3.2.9.3	Slice data partition C RBSP syntax	45
7.3.2.10	RBSP slice trailing bits syntax	45
7.3.2.11	RBSP trailing bits syntax	45
7.3.3	Slice header syntax	46
7.3.3.1	Reference picture list reordering syntax	47
7.3.3.2	Prediction weight table syntax	48
7.3.3.3	Decoded reference picture marking syntax	49
7.3.4	Slice data syntax	50
7.3.5	Macroblock layer syntax	51
7.3.5.1	Macroblock prediction syntax	52
7.3.5.2	Sub-macroblock prediction syntax	53
7.3.5.3	Residual data syntax	54
7.3.5.3.1	Residual luma syntax	55
7.3.5.3.2	Residual block CAVLC syntax	56
7.3.5.3.3	Residual block CABAC syntax	57
7.4	<i>Semantics</i>	58
7.4.1	NAL unit semantics	58
7.4.1.1	Encapsulation of an SODB within an RBSP (informative)	61
7.4.1.2	Order of NAL units and association to coded pictures, access units, and video sequences	61
7.4.1.2.1	Order of sequence and picture parameter set RBSPs and their activation	61
7.4.1.2.2	Order of access units and association to coded video sequences	62
7.4.1.2.3	Order of NAL units and coded pictures and association to access units	63
7.4.1.2.4	Detection of the first VCL NAL unit of a primary coded picture	65
7.4.1.2.5	Order of VCL NAL units and association to coded pictures	65
7.4.2	Raw byte sequence payloads and RBSP trailing bits semantics	66
7.4.2.1	Sequence parameter set RBSP semantics	66
7.4.2.1.1	Sequence parameter set data semantics	66
7.4.2.1.1.1	Scaling list semantics	72
7.4.2.1.2	Sequence parameter set extension RBSP semantics	72
7.4.2.2	Picture parameter set RBSP semantics	74
7.4.2.3	Supplemental enhancement information RBSP semantics	77
7.4.2.3.1	Supplemental enhancement information message semantics	77
7.4.2.4	Access unit delimiter RBSP semantics	77
7.4.2.5	End of sequence RBSP semantics	77
7.4.2.6	End of stream RBSP semantics	77
7.4.2.7	Filler data RBSP semantics	77
7.4.2.8	Slice layer without partitioning RBSP semantics	77
7.4.2.9	Slice data partition RBSP semantics	78
7.4.2.9.1	Slice data partition A RBSP semantics	78
7.4.2.9.2	Slice data partition B RBSP semantics	78
7.4.2.9.3	Slice data partition C RBSP semantics	78
7.4.2.10	RBSP slice trailing bits semantics	79
7.4.2.11	RBSP trailing bits semantics	79
7.4.3	Slice header semantics	79
7.4.3.1	Reference picture list reordering semantics	85

7.4.3.2	Prediction weight table semantics	86
7.4.3.3	Decoded reference picture marking semantics	86
7.4.4	Slice data semantics	89
7.4.5	Macroblock layer semantics	90
7.4.5.1	Macroblock prediction semantics	97
7.4.5.2	Sub-macroblock prediction semantics	97
7.4.5.3	Residual data semantics	100
7.4.5.3.1	Residual luma data semantics	100
7.4.5.3.2	Residual block CAVLC semantics	101
7.4.5.3.3	Residual block CABAC semantics	101
8	Decoding process	102
8.1	NAL unit decoding process	103
8.2	Slice decoding process	104
8.2.1	Decoding process for picture order count	104
8.2.1.1	Decoding process for picture order count type 0	105
8.2.1.2	Decoding process for picture order count type 1	106
8.2.1.3	Decoding process for picture order count type 2	107
8.2.2	Decoding process for macroblock to slice group map	108
8.2.2.1	Specification for interleaved slice group map type	109
8.2.2.2	Specification for dispersed slice group map type	109
8.2.2.3	Specification for foreground with left-over slice group map type	109
8.2.2.4	Specification for box-out slice group map types	110
8.2.2.5	Specification for raster scan slice group map types	110
8.2.2.6	Specification for wipe slice group map types	111
8.2.2.7	Specification for explicit slice group map type	111
8.2.2.8	Specification for conversion of map unit to slice group map to macroblock to slice group map	111
8.2.3	Decoding process for slice data partitioning	111
8.2.4	Decoding process for reference picture lists construction	112
8.2.4.1	Decoding process for picture numbers	112
8.2.4.2	Initialisation process for reference picture lists	113
8.2.4.2.1	Initialisation process for the reference picture list for P and SP slices in frames	114
8.2.4.2.2	Initialisation process for the reference picture list for P and SP slices in fields	114
8.2.4.2.3	Initialisation process for reference picture lists for B slices in frames	115
8.2.4.2.4	Initialisation process for reference picture lists for B slices in fields	115
8.2.4.2.5	Initialisation process for reference picture lists in fields	116
8.2.4.3	Reordering process for reference picture lists	116
8.2.4.3.1	Reordering process of reference picture lists for short-term reference pictures	117
8.2.4.3.2	Reordering process of reference picture lists for long-term reference pictures	118
8.2.5	Decoded reference picture marking process	118
8.2.5.1	Sequence of operations for decoded reference picture marking process	119
8.2.5.2	Decoding process for gaps in frame_num	119
8.2.5.3	Sliding window decoded reference picture marking process	120
8.2.5.4	Adaptive memory control decoded reference picture marking process	120
8.2.5.4.1	Marking process of a short-term reference picture as “unused for reference”	120
8.2.5.4.2	Marking process of a long-term reference picture as “unused for reference”	121
8.2.5.4.3	Assignment process of a LongTermFrameIdx to a short-term reference picture	121
8.2.5.4.4	Decoding process for MaxLongTermFrameIdx	121
8.2.5.4.4.1	Marking process of all reference pictures as “unused for reference” and setting MaxLongTermFrameIdx to “no long-term frame indices”	121
8.2.5.4.5	Process for assigning a long-term frame index to the current picture	122
8.3	Intra prediction process	122
8.3.1	Intra_4x4 prediction process for luma samples	123
8.3.1.1	Derivation process for the Intra4x4PredMode	123
8.3.1.2	Intra_4x4 sample prediction	125
8.3.1.2.1	Specification of Intra_4x4_Veritical prediction mode	126
8.3.1.2.2	Specification of Intra_4x4_Horizontal prediction mode	126
8.3.1.2.3	Specification of Intra_4x4_DC prediction mode	126
8.3.1.2.4	Specification of Intra_4x4_Diagonal_Down_Left prediction mode	127
8.3.1.2.5	Specification of Intra_4x4_Diagonal_Down_Right prediction mode	127
8.3.1.2.6	Specification of Intra_4x4_Veritical_Right prediction mode	127
8.3.1.2.7	Specification of Intra_4x4_Horizontal_Down prediction mode	128
8.3.1.2.8	Specification of Intra_4x4_Veritical_Left prediction mode	128
8.3.1.2.9	Specification of Intra_4x4_Horizontal_Up prediction mode	128

8.3.2	Intra_8x8 prediction process for luma samples	129
8.3.2.1	Derivation process for Intra8x8PredMode.....	129
8.3.2.2	Intra_8x8 sample prediction	131
8.3.2.2.1	Reference sample filtering process for Intra_8x8 sample prediction	132
8.3.2.2.2	Specification of Intra_8x8_Veritical prediction mode	133
8.3.2.2.3	Specification of Intra_8x8_Horizontal prediction mode	133
8.3.2.2.4	Specification of Intra_8x8_DC prediction mode	133
8.3.2.2.5	Specification of Intra_8x8_Diagonal_Down_Left prediction mode	134
8.3.2.2.6	Specification of Intra_8x8_Diagonal_Down_Right prediction mode	134
8.3.2.2.7	Specification of Intra_8x8_Veritical_Right prediction mode	135
8.3.2.2.8	Specification of Intra_8x8_Horizontal_Down prediction mode	135
8.3.2.2.9	Specification of Intra_8x8_Veritical_Left prediction mode	135
8.3.2.2.10	Specification of Intra_8x8_Horizontal_Up prediction mode	136
8.3.3	Intra_16x16 prediction process for luma samples.....	136
8.3.3.1	Specification of Intra_16x16_Veritical prediction mode	137
8.3.3.2	Specification of Intra_16x16_Horizontal prediction mode.....	137
8.3.3.3	Specification of Intra_16x16_DC prediction mode	137
8.3.3.4	Specification of Intra_16x16_Plane prediction mode	138
8.3.4	Intra prediction process for chroma samples	138
8.3.4.1	Specification of Intra_Chroma_DC prediction mode	140
8.3.4.2	Specification of Intra_Chroma_Horizontal prediction mode.....	141
8.3.4.3	Specification of Intra_Chroma_Veritical prediction mode	141
8.3.4.4	Specification of Intra_Chroma_Plane prediction mode	142
8.3.4.5	Intra prediction for chroma samples with ChromaArrayType equal to 3	142
8.3.5	Sample construction process for I_PCM macroblocks	143
8.4	<i>Inter prediction process</i>	143
8.4.1	Derivation process for motion vector components and reference indices.....	146
8.4.1.1	Derivation process for luma motion vectors for skipped macroblocks in P and SP slices.....	147
8.4.1.2	Derivation process for luma motion vectors for B_Skip, B_Direct_16x16, and B_Direct_8x8.....	147
8.4.1.2.1	Derivation process for the co-located 4x4 sub-macroblock partitions	148
8.4.1.2.2	Derivation process for spatial direct luma motion vector and reference index prediction mode	151
8.4.1.2.3	Derivation process for temporal direct luma motion vector and reference index prediction mode	152
8.4.1.3	Derivation process for luma motion vector prediction.....	155
8.4.1.3.1	Derivation process for median luma motion vector prediction	156
8.4.1.3.2	Derivation process for motion data of neighbouring partitions.....	157
8.4.1.4	Derivation process for chroma motion vectors	158
8.4.2	Decoding process for Inter prediction samples	158
8.4.2.1	Reference picture selection process	159
8.4.2.2	Fractional sample interpolation process.....	160
8.4.2.2.1	Luma sample interpolation process.....	162
8.4.2.2.2	Chroma sample interpolation process	164
8.4.2.3	Weighted sample prediction process.....	165
8.4.2.3.1	Default weighted sample prediction process.....	166
8.4.2.3.2	Weighted sample prediction process.....	166
8.4.3	Derivation process for prediction weights	167
8.5	<i>Transform coefficient decoding process and picture construction process prior to deblocking filter process.</i> 169	
8.5.1	Specification of transform decoding process for 4x4 luma residual blocks.....	169
8.5.2	Specification of transform decoding process for luma samples of Intra_16x16 macroblock prediction mode	170
8.5.3	Specification of transform decoding process for 8x8 luma residual blocks.....	171
8.5.4	Specification of transform decoding process for chroma samples.....	171
8.5.5	Specification of transform decoding process for chroma samples with ChromaArrayType equal to 3	173
8.5.6	Inverse scanning process for transform coefficients	174
8.5.7	Inverse scanning process for 8x8 transform coefficients	174
8.5.8	Derivation process for the chroma quantisation parameters and scaling function	176
8.5.9	Scaling and transformation process for DC transform coefficients for Intra_16x16 macroblock type... ..	178
8.5.10	Scaling and transformation process for chroma DC transform coefficients	179
8.5.10.1	Transformation process for chroma DC transform coefficients	179
8.5.10.2	Scaling process for chroma DC transform coefficients	179
8.5.11	Scaling and transformation process for residual 4x4 blocks.....	180
8.5.11.1	Scaling process for residual 4x4 blocks	181
8.5.11.2	Transformation process for residual 4x4 blocks	182

8.5.12	Scaling and transformation process for residual 8x8 blocks	183
8.5.12.1	Scaling process for residual 8x8 blocks	183
8.5.12.2	Transformation process for residual 8x8 blocks	184
8.5.13	Picture construction process prior to deblocking filter process	187
8.5.14	Intra residual transform-bypass decoding process	188
8.6	<i>Decoding process for P macroblocks in SP slices or SI macroblocks</i>	189
8.6.1	SP decoding process for non-switching pictures	189
8.6.1.1	Luma transform coefficient decoding process	189
8.6.1.2	Chroma transform coefficient decoding process	190
8.6.2	SP and SI slice decoding process for switching pictures	192
8.6.2.1	Luma transform coefficient decoding process	192
8.6.2.2	Chroma transform coefficient decoding process	192
8.7	<i>Deblocking filter process</i>	193
8.7.1	Filtering process for block edges	197
8.7.2	Filtering process for a set of samples across a horizontal or vertical block edge	199
8.7.2.1	Derivation process for the luma content dependent boundary filtering strength	200
8.7.2.2	Derivation process for the thresholds for each block edge	201
8.7.2.3	Filtering process for edges with bS less than 4	202
8.7.2.4	Filtering process for edges for bS equal to 4	204
9	Parsing process	205
9.1	<i>Parsing process for Exp-Golomb codes</i>	205
9.1.1	Mapping process for signed Exp-Golomb codes	207
9.1.2	Mapping process for coded block pattern	207
9.2	<i>CAVLC parsing process for transform coefficient levels</i>	210
9.2.1	Parsing process for total number of transform coefficient levels and trailing ones	210
9.2.2	Parsing process for level information	213
9.2.2.1	Parsing process for level_prefix	214
9.2.3	Parsing process for run information	215
9.2.4	Combining level and run information	218
9.3	<i>CABAC parsing process for slice data</i>	218
9.3.1	Initialisation process	219
9.3.1.1	Initialisation process for context variables	220
9.3.1.2	Initialisation process for the arithmetic decoding engine	242
9.3.2	Binarization process	242
9.3.2.1	Unary (U) binarization process	245
9.3.2.2	Truncated unary (TU) binarization process	246
9.3.2.3	Concatenated unary/ k-th order Exp-Golomb (UEGk) binarization process	246
9.3.2.4	Fixed-length (FL) binarization process	247
9.3.2.5	Binarization process for macroblock type and sub-macroblock type	247
9.3.2.6	Binarization process for coded block pattern	250
9.3.2.7	Binarization process for mb_qp_delta	250
9.3.3	Decoding process flow	250
9.3.3.1	Derivation process for ctxIdx	251
9.3.3.1.1	Assignment process of ctxIdxInc using neighbouring syntax elements	253
9.3.3.1.1.1	Derivation process of ctxIdxInc for the syntax element mb_skip_flag	253
9.3.3.1.1.2	Derivation process of ctxIdxInc for the syntax element mb_field_decoding_flag	253
9.3.3.1.1.3	Derivation process of ctxIdxInc for the syntax element mb_type	254
9.3.3.1.1.4	Derivation process of ctxIdxInc for the syntax element coded_block_pattern	254
9.3.3.1.1.5	Derivation process of ctxIdxInc for the syntax element mb_qp_delta	255
9.3.3.1.1.6	Derivation process of ctxIdxInc for the syntax elements ref_idx_10 and ref_idx_11	255
9.3.3.1.1.7	Derivation process of ctxIdxInc for the syntax elements mvd_10 and mvd_11	256
9.3.3.1.1.8	Derivation process of ctxIdxInc for the syntax element intra_chroma_pred_mode	257
9.3.3.1.1.9	Derivation process of ctxIdxInc for the syntax element coded_block_flag	258
9.3.3.1.1.10	Derivation process of ctxIdxInc for the syntax element transform_size_8x8_flag	261
9.3.3.1.2	Assignment process of ctxIdxInc using prior decoded bin values	261
9.3.3.1.3	Assignment process of ctxIdxInc for syntax elements significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1	261
9.3.3.2	Arithmetic decoding process	264
9.3.3.2.1	Arithmetic decoding process for a binary decision	265
9.3.3.2.1.1	State transition process	266
9.3.3.2.2	Renormalization process in the arithmetic decoding engine	268
9.3.3.2.3	Bypass decoding process for binary decisions	269
9.3.3.2.4	Decoding process for binary decisions before termination	269

9.3.4	Arithmetic encoding process (informative)	270
9.3.4.1	Initialisation process for the arithmetic encoding engine (informative)	270
9.3.4.2	Encoding process for a binary decision (informative)	270
9.3.4.3	Renormalization process in the arithmetic encoding engine (informative).....	271
9.3.4.4	Bypass encoding process for binary decisions (informative).....	273
9.3.4.5	Encoding process for a binary decision before termination (informative).....	273
9.3.4.6	Byte stuffing process (informative)	274
Annex A	Profiles and levels.....	276
A.1	Requirements on video decoder capability.....	276
A.2	Profiles.....	276
A.2.1	Baseline profile	276
A.2.2	Main profile	277
A.2.3	Extended profile.....	277
A.2.4	High profile.....	277
A.2.5	High 10 profile.....	278
A.2.6	High 4:2:2 profile.....	278
A.2.7	High 4:4:4 Predictive profile	279
A.2.8	High 10 Intra profile	279
A.2.9	High 4:2:2 Intra profile	280
A.2.10	High 4:4:4 Intra profile	280
A.2.11	CAVLC 4:4:4 Intra profile.....	281
A.3	Levels	281
A.3.1	Level limits common to the Baseline, Main, and Extended profiles.....	281
A.3.2	Level limits common to the High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profiles	283
A.3.3	Profile-specific level limits	284
A.3.3.1	Baseline profile level limits.....	286
A.3.3.2	Main, High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profile level limits	286
A.3.3.3	Extended Profile level limits	287
A.3.4	Effect of level limits on frame rate (informative)	288
Annex B	Byte stream format	291
B.1	Byte stream NAL unit syntax and semantics	291
B.1.1	Byte stream NAL unit syntax	291
B.1.2	Byte stream NAL unit semantics	291
B.2	Byte stream NAL unit decoding process	291
B.3	Decoder byte-alignment recovery (informative).....	292
Annex C	Hypothetical reference decoder	293
C.1	Operation of coded picture buffer (CPB)	295
C.1.1	Timing of bitstream arrival	295
C.1.2	Timing of coded picture removal.....	296
C.2	Operation of the decoded picture buffer (DPB).....	297
C.2.1	Decoding of gaps in frame_num and storage of "non-existing" frames.....	297
C.2.2	Picture decoding and output.....	297
C.2.3	Removal of pictures from the DPB before possible insertion of the current picture	298
C.2.4	Current decoded picture marking and storage	298
C.2.4.1	Marking and storage of a reference decoded picture into the DPB	298
C.2.4.2	Storage of a non-reference picture into the DPB.....	299
C.3	Bitstream conformance.....	299
C.4	Decoder conformance.....	300
C.4.1	Operation of the output order DPB	301
C.4.2	Decoding of gaps in frame_num and storage of "non-existing" pictures.....	301
C.4.3	Picture decoding	301
C.4.4	Removal of pictures from the DPB before possible insertion of the current picture	302
C.4.5	Current decoded picture marking and storage	302
C.4.5.1	Storage and marking of a reference decoded picture into the DPB	302
C.4.5.2	Storage and marking of a non-reference decoded picture into the DPB.....	303
C.4.5.3	"Bumping" process	303
Annex D	Supplemental enhancement information	305
D.1	SEI payload syntax	305
D.1.1	Buffering period SEI message syntax.....	307
D.1.2	Picture timing SEI message syntax.....	307

D.1.3	Pan-scan rectangle SEI message syntax.....	308
D.1.4	Filler payload SEI message syntax	308
D.1.5	User data registered by ITU-T Rec. T.35 SEI message syntax	309
D.1.6	User data unregistered SEI message syntax.....	309
D.1.7	Recovery point SEI message syntax	309
D.1.8	Decoded reference picture marking repetition SEI message syntax	309
D.1.9	Spare picture SEI message syntax	310
D.1.10	Scene information SEI message syntax	310
D.1.11	Sub-sequence information SEI message syntax.....	311
D.1.12	Sub-sequence layer characteristics SEI message syntax.....	311
D.1.13	Sub-sequence characteristics SEI message syntax.....	311
D.1.14	Full-frame freeze SEI message syntax.....	312
D.1.15	Full-frame freeze release SEI message syntax.....	312
D.1.16	Full-frame snapshot SEI message syntax.....	312
D.1.17	Progressive refinement segment start SEI message syntax.....	312
D.1.18	Progressive refinement segment end SEI message syntax.....	312
D.1.19	Motion-constrained slice group set SEI message syntax	312
D.1.20	Film grain characteristics SEI message syntax	313
D.1.21	Deblocking filter display preference SEI message syntax	313
D.1.22	Stereo video information SEI message syntax.....	314
D.1.23	Post-filter hint SEI message syntax	314
D.1.24	Tone mapping information SEI message syntax.....	315
D.1.25	Reserved SEI message syntax.....	315
D.2	<i>SEI payload semantics.....</i>	<i>315</i>
D.2.1	Buffering period SEI message semantics.....	315
D.2.2	Picture timing SEI message semantics.....	316
D.2.3	Pan-scan rectangle SEI message semantics	319
D.2.4	Filler payload SEI message semantics	321
D.2.5	User data registered by ITU-T Rec. T.35 SEI message semantics.....	321
D.2.6	User data unregistered SEI message semantics	321
D.2.7	Recovery point SEI message semantics.....	321
D.2.8	Decoded reference picture marking repetition SEI message semantics	323
D.2.9	Spare picture SEI message semantics	323
D.2.10	Scene information SEI message semantics.....	324
D.2.11	Sub-sequence information SEI message semantics	326
D.2.12	Sub-sequence layer characteristics SEI message semantics.....	328
D.2.13	Sub-sequence characteristics SEI message semantics	328
D.2.14	Full-frame freeze SEI message semantics.....	330
D.2.15	Full-frame freeze release SEI message semantics.....	330
D.2.16	Full-frame snapshot SEI message semantics	330
D.2.17	Progressive refinement segment start SEI message semantics	331
D.2.18	Progressive refinement segment end SEI message semantics.....	331
D.2.19	Motion-constrained slice group set SEI message semantics	331
D.2.20	Film grain characteristics SEI message semantics.....	332
D.2.21	Deblocking filter display preference SEI message semantics.....	338
D.2.22	Stereo video information SEI message semantics	339
D.2.23	Post-filter hint SEI message semantics	340
D.2.24	Tone mapping information SEI message semantics	341
D.2.25	Reserved SEI message semantics	343
Annex E	Video usability information.....	344
E.1	<i>VUI syntax</i>	<i>344</i>
E.1.1	VUI parameters syntax	344
E.1.2	HRD parameters syntax	345
E.2	<i>VUI semantics.....</i>	<i>346</i>
E.2.1	VUI parameters semantics	346
E.2.2	HRD parameters semantics.....	357
Annex G	Scalable video coding.....	359
G.1	<i>Scope.....</i>	<i>359</i>
G.2	<i>Normative References</i>	<i>359</i>
G.3	<i>Definitions</i>	<i>359</i>
G.4	<i>Abbreviations.....</i>	<i>363</i>
G.5	<i>Conventions</i>	<i>363</i>

<i>G.6 Source, coded, decoded and output data formats, scanning processes, neighbouring and reference layer relationships</i>	363
G.6.1 Derivation process for reference layer macroblocks	363
G.6.1.1 Field-to-frame reference layer macroblock conversion process	365
G.6.1.2 Frame-to-field reference layer macroblock conversion process	366
G.6.2 Derivation process for reference layer partitions	366
G.6.3 Derivation process for reference layer sample locations in resampling	367
<i>G.7 Syntax and semantics</i>	369
G.7.1 Method of specifying syntax in tabular form	369
G.7.2 Specification of syntax functions, categories, and descriptors	369
G.7.3 Syntax in tabular form	370
G.7.3.1 NAL unit syntax	370
G.7.3.1.1 NAL unit header SVC extension syntax	370
G.7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax	370
G.7.3.2.1 Sequence parameter set RBSP syntax	370
G.7.3.2.1.1 Sequence parameter set data syntax	370
G.7.3.2.1.2 Sequence parameter set extension RBSP syntax	370
G.7.3.2.1.3 Subset sequence parameter set RBSP syntax	371
G.7.3.2.1.4 Sequence parameter set SVC extension syntax	371
G.7.3.2.2 Picture parameter set RBSP syntax	371
G.7.3.2.3 Supplemental enhancement information RBSP syntax	372
G.7.3.2.3.1 Supplemental enhancement information message syntax	372
G.7.3.2.4 Access unit delimiter RBSP syntax	372
G.7.3.2.5 End of sequence RBSP syntax	372
G.7.3.2.6 End of stream RBSP syntax	372
G.7.3.2.7 Filler data RBSP syntax	372
G.7.3.2.8 Slice layer without partitioning RBSP syntax	372
G.7.3.2.9 Slice data partition RBSP syntax	372
G.7.3.2.10 RBSP slice trailing bits syntax	372
G.7.3.2.11 RBSP trailing bits syntax	372
G.7.3.2.12 Prefix NAL unit RBSP syntax	372
G.7.3.2.13 Slice layer in scalable extension RBSP syntax	373
G.7.3.3 Slice header syntax	373
G.7.3.3.1 Reference picture list reordering syntax	373
G.7.3.3.2 Prediction weight table syntax	373
G.7.3.3.3 Decoded reference picture marking syntax	373
G.7.3.3.4 Slice header in scalable extension syntax	373
G.7.3.3.5 Decoded reference base picture marking syntax	376
G.7.3.4 Slice data syntax	376
G.7.3.4.1 Slice data in scalable extension syntax	376
G.7.3.5 Macroblock layer syntax	377
G.7.3.5.1 Macroblock prediction syntax	377
G.7.3.5.2 Sub-macroblock prediction syntax	377
G.7.3.5.3 Residual data syntax	377
G.7.3.5.3.1 Residual luma syntax	377
G.7.3.5.3.2 Residual block CAVLC syntax	377
G.7.3.5.3.3 Residual block CABAC syntax	377
G.7.3.6 Macroblock layer in scalable extension syntax	377
G.7.3.6.1 Macroblock prediction in scalable extension syntax	379
G.7.3.6.2 Sub-macroblock prediction in scalable extension syntax	380
G.7.4 Semantics	380
G.7.4.1 NAL unit semantics	381
G.7.4.1.1 NAL unit header SVC extension semantics	381
G.7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences	382
G.7.4.1.2.1 Order of SVC sequence parameter set RBSPs and picture parameter set RBSPs and their activation	382
G.7.4.1.2.2 Order of access units and association to coded video sequences	386
G.7.4.1.2.3 Order of NAL units and coded pictures and association to access units	387
G.7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture	387
G.7.4.1.2.5 Order of VCL NAL units and association to coded pictures	388
G.7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics	388
G.7.4.2.1 Sequence parameter set RBSP semantics	388
G.7.4.2.1.1 Sequence parameter set data semantics	388

G.7.4.2.1.2	Sequence parameter set extension RBSP semantics.....	389
G.7.4.2.1.3	Subset sequence parameter set RBSP semantics.....	389
G.7.4.2.1.4	Sequence parameter set SVC extension semantics.....	389
G.7.4.2.2	Picture parameter set RBSP semantics.....	391
G.7.4.2.3	Supplemental enhancement information RBSP semantics.....	391
G.7.4.2.3.1	Supplemental enhancement information message semantics.....	391
G.7.4.2.4	Access unit delimiter RBSP semantics.....	391
G.7.4.2.5	End of sequence RBSP semantics.....	391
G.7.4.2.6	End of stream RBSP semantics.....	391
G.7.4.2.7	Filler data RBSP semantics.....	391
G.7.4.2.8	Slice layer without partitioning RBSP semantics.....	391
G.7.4.2.9	Slice data partition RBSP semantics.....	391
G.7.4.2.10	RBSP slice trailing bits semantics.....	391
G.7.4.2.11	RBSP trailing bits semantics.....	391
G.7.4.2.12	Prefix NAL unit RBSP semantics.....	391
G.7.4.2.13	Slice layer in scalable extension RBSP semantics.....	392
G.7.4.3	Slice header semantics.....	392
G.7.4.3.1	Reference picture list reordering semantics.....	392
G.7.4.3.2	Prediction weight table semantics.....	392
G.7.4.3.3	Decoded reference picture marking semantics.....	392
G.7.4.3.4	Slice header in scalable extension semantics.....	392
G.7.4.3.5	Decoded reference base picture marking semantics.....	401
G.7.4.4	Slice data semantics.....	402
G.7.4.4.1	Slice data in scalable extension semantics.....	402
G.7.4.5	Macroblock layer semantics.....	402
G.7.4.5.1	Macroblock prediction semantics.....	402
G.7.4.5.2	Sub-macroblock prediction semantics.....	402
G.7.4.5.3	Residual data semantics.....	402
G.7.4.5.3.1	Residual luma semantics.....	402
G.7.4.5.3.2	Residual block CAVLC semantics.....	402
G.7.4.5.3.3	Residual block CABAC semantics.....	402
G.7.4.6	Macroblock layer in scalable extension semantics.....	403
G.7.4.6.1	Macroblock prediction in scalable extension semantics.....	404
G.7.4.6.2	Sub-macroblock prediction in scalable extension semantics.....	404
G.8	<i>SVC decoding process</i>	405
G.8.1	SVC initialisation and decoding processes.....	406
G.8.1.1	Derivation process for the set of layer representations required for decoding.....	406
G.8.1.2	Array assignment, initialisation, and restructuring processes.....	406
G.8.1.2.1	Array assignment and initialisation process.....	406
G.8.1.2.2	Array restructuring process.....	408
G.8.1.3	Layer representation decoding processes.....	409
G.8.1.3.1	Base decoding process for layer representations without resolution change.....	409
G.8.1.3.2	Base decoding process for layer representations with resolution change.....	410
G.8.1.3.3	Target layer representation decoding process.....	411
G.8.1.4	Slice decoding processes.....	412
G.8.1.4.1	Base decoding process for slices without resolution change.....	412
G.8.1.4.2	Base decoding process for slices with resolution change.....	412
G.8.1.5	Macroblock initialisation and decoding processes.....	413
G.8.1.5.1	Macroblock initialisation process.....	413
G.8.1.5.1.1	Derivation process for macroblock type, sub-macroblock type, and inter-layer predictors for reference indices and motion vectors.....	414
G.8.1.5.1.2	Derivation process for quantisation parameters and transform type.....	415
G.8.1.5.2	Base decoding process for macroblocks in slices without resolution change.....	417
G.8.1.5.3	Base decoding process for macroblocks in slices with resolution change.....	418
G.8.1.5.4	Macroblock decoding process prior to decoding a layer representation without resolution change and tcoeff_level_prediction_flag equal to 0.....	420
G.8.1.5.5	Macroblock decoding process prior to resolution change.....	420
G.8.1.5.6	Target macroblock decoding process.....	421
G.8.2	SVC reference picture lists construction and decoded reference picture marking process.....	423
G.8.2.1	SVC decoding process for picture order count.....	424
G.8.2.2	SVC decoding process for picture numbers.....	425
G.8.2.3	SVC decoding process for reference picture lists construction.....	427
G.8.2.4	SVC decoded reference picture marking process.....	428

G.8.2.4.1	SVC reference picture marking process for a dependency representation	428
G.8.2.5	SVC decoding process for gaps in frame_num	430
G.8.3	SVC intra decoding processes	431
G.8.3.1	SVC derivation process for intra prediction modes	431
G.8.3.1.1	SVC derivation process for Intra_4x4 prediction modes	432
G.8.3.1.2	SVC derivation process for Intra_8x8 prediction modes	432
G.8.3.2	SVC intra sample prediction and construction process	434
G.8.3.2.1	SVC intra prediction and construction process for luma samples or chroma samples with ChromaArrayType equal to 3	435
G.8.3.2.1.1	SVC construction process for luma samples and chroma samples with ChromaArrayType equal to 3 of I_PCM macroblocks	435
G.8.3.2.1.2	SVC Intra_4x4 sample prediction and construction process	436
G.8.3.2.1.3	SVC Intra_8x8 sample prediction and construction process	436
G.8.3.2.1.4	SVC Intra_16x16 sample prediction and construction process	437
G.8.3.2.2	SVC intra prediction and construction process for chroma samples	438
G.8.3.2.2.1	SVC construction process for chroma samples of I_PCM macroblocks	438
G.8.3.2.2.2	SVC intra prediction and construction process for chroma samples with ChromaArrayType equal to 1 or 2	438
G.8.4	SVC Inter prediction process	439
G.8.4.1	SVC derivation process for motion vector components and reference indices	439
G.8.4.1.1	SVC derivation process for luma motion vector components and reference indices of a macroblock or sub-macroblock partition	441
G.8.4.1.2	SVC derivation process for luma motion vectors and reference indices for B_Skip, B_Direct_16x16, and B_Direct_8x8 in NAL units with nal_unit_type equal to 20	444
G.8.4.2	SVC decoding process for Inter prediction samples	445
G.8.4.2.1	SVC derivation process for prediction weights	448
G.8.4.2.2	Intra-inter prediction combination process	449
G.8.5	SVC transform coefficient decoding and sample array construction processes	450
G.8.5.1	Transform coefficient scaling and refinement process	450
G.8.5.1.1	Refinement process for luma transform coefficients or chroma transform coefficients with ChromaArrayType equal to 3	451
G.8.5.1.1.1	Assignment process for luma transform coefficient values or chroma transform coefficient values with ChromaArrayType equal to 3 for I_PCM macroblocks	451
G.8.5.1.1.2	Refinement process for transform coefficients of residual 4x4 blocks	452
G.8.5.1.1.3	Refinement process for transform coefficients of residual 8x8 blocks	453
G.8.5.1.1.4	Refinement process for transform coefficients of Intra_16x16 macroblocks	453
G.8.5.1.2	Refinement process for chroma transform coefficients	455
G.8.5.1.2.1	Assignment process for chroma transform coefficient values for I_PCM macroblocks	455
G.8.5.1.2.2	Refinement process for chroma transform coefficients with ChromaArrayType equal to 1 or 2	456
G.8.5.2	Transform coefficient level scaling process prior to transform coefficient refinement	457
G.8.5.3	Residual construction and accumulation process	459
G.8.5.3.1	Construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3	460
G.8.5.3.1.1	Construction process for luma residuals or chroma residuals with ChromaArrayType equal to 3 of I_PCM macroblocks	460
G.8.5.3.1.2	Construction process for residual 4x4 blocks	460
G.8.5.3.1.3	Construction process for residual 8x8 blocks	461
G.8.5.3.1.4	Construction process for residuals of Intra_16x16 macroblocks	461
G.8.5.3.2	Construction process for chroma residuals	462
G.8.5.3.2.1	Construction process for chroma residuals of I_PCM macroblocks	462
G.8.5.3.2.2	Construction process for chroma residuals with ChromaArrayType equal to 1 or 2	462
G.8.5.4	Sample array accumulation process	463
G.8.5.4.1	Picture sample array construction process	464
G.8.5.4.2	Macroblock sample array extraction process	464
G.8.5.4.3	Picture sample array construction process for a colour component	465
G.8.5.4.4	Macroblock sample array extraction process for a colour component	465
G.8.5.5	Sample array re-initialisation process	466
G.8.6	Resampling processes for prediction data, intra samples, and residual samples	466
G.8.6.1	Derivation process for inter-layer predictors for macroblock type, sub-macroblock type, reference indices, and motion vectors	466
G.8.6.1.1	Derivation process for reference layer partition identifications	467
G.8.6.1.2	Derivation process for inter-layer predictors for reference indices and motion vectors	469

G.8.6.1.3	Derivation process for inter-layer predictors for P and B macroblock and sub-macroblock types	473
G.8.6.2	Resampling process for intra samples	476
G.8.6.2.1	Resampling process for intra samples of a macroblock colour component	476
G.8.6.2.2	Reference layer sample array construction process prior to intra resampling	478
G.8.6.2.2.1	Derivation process for reference layer slice and intra macroblock identifications	480
G.8.6.2.2.2	Construction process for not available sample values prior to intra resampling	481
G.8.6.2.3	Interpolation process for Intra_Base prediction	482
G.8.6.2.4	Vertical interpolation process for Intra_Base prediction	484
G.8.6.3	Resampling process for residual samples	485
G.8.6.3.1	Resampling process for residual samples of a macroblock colour component	486
G.8.6.3.2	Reference layer sample array construction process prior to residual resampling	487
G.8.6.3.2.1	Derivation process for reference layer transform block identifications	489
G.8.6.3.3	Interpolation process for residual prediction	490
G.8.6.3.4	Vertical interpolation process for residual prediction	491
G.8.7	SVC deblocking filter processes	491
G.8.7.1	Deblocking filter process for Intra_Base prediction	491
G.8.7.2	Deblocking filter process for target representations	492
G.8.7.3	Derivation process for quantisation parameters used in the deblocking filter process	492
G.8.7.4	Macroblock deblocking filter process	493
G.8.7.4.1	SVC filtering process for block edges	497
G.8.7.4.2	SVC filtering process for a set of samples across a horizontal or vertical block edge	498
G.8.7.4.3	SVC derivation process for the luma content dependent boundary filtering strength	499
G.8.8	Specification of bitstream subsets	502
G.8.8.1	Sub-bitstream extraction process	502
G.8.8.2	Specification of the base layer bitstream	502
G.9	<i>Parsing process</i>	502
G.9.1	Alternative parsing process for coded block pattern	503
G.9.2	Alternative CAVLC parsing process for transform coefficient levels	504
G.9.2.1	Additional parsing process for total number of transform coefficient levels and trailing ones	504
G.9.2.2	Alternative parsing process for run information	506
G.9.3	Alternative CABAC parsing process for slice data in scalable extension	507
G.9.3.1	Initialisation process	507
G.9.3.2	Binarization process	509
G.9.3.3	Decoding process flow	509
G.9.3.3.1	Derivation process for ctxIdx	509
G.9.3.3.2	Assignment process of ctxIdxInc using neighbouring syntax elements	510
G.9.3.3.2.1	Derivation process of ctxIdxInc for the syntax element base_mode_flag	510
G.9.3.3.2.2	Derivation process of ctxIdxInc for the syntax element residual_prediction_flag	510
G.10	<i>Profiles and levels</i>	510
G.10.1	Profiles	510
G.10.1.1	Scalable Baseline profile	510
G.10.1.2	Scalable High profile	511
G.10.1.3	Scalable High Intra profile	512
G.10.2	Levels	513
G.10.2.1	Level limits common to Scalable Baseline, Scalable High, and Scalable High Intra profiles	513
G.10.2.2	Profile specific level limits	515
G.11	<i>Byte stream format</i>	517
G.12	<i>Hypothetical reference decoder</i>	517
G.13	<i>Supplemental enhancement information</i>	517
G.13.1	SEI payload syntax	517
G.13.1.1	Scalability information SEI message syntax	517
G.13.1.2	Layers not present SEI message syntax	520
G.13.1.3	Layer dependency change SEI message syntax	520
G.13.1.4	Sub-picture scalable layer SEI message syntax	520
G.13.1.5	Non-required layer representation SEI message syntax	521
G.13.1.6	Priority layer information SEI message syntax	521
G.13.1.7	Scalable nesting SEI message syntax	521
G.13.1.8	Base layer temporal HRD SEI message syntax	522
G.13.1.9	Quality layer integrity check SEI message syntax	522
G.13.1.10	Redundant picture property SEI message syntax	523
G.13.1.11	Temporal level zero dependency representation index SEI message syntax	523
G.13.1.12	Temporal level switching point SEI message syntax	523

G.13.2	SEI payload semantics	523
G.13.2.1	Scalability information SEI message semantics	524
G.13.2.2	Layers not present SEI message semantics	532
G.13.2.3	Layer dependency change SEI message semantics	533
G.13.2.4	Sub-picture scalable layer SEI message semantics	533
G.13.2.5	Non-required layer representation SEI message semantics	533
G.13.2.6	Priority layer information SEI message semantics	534
G.13.2.7	Scalable nesting SEI message semantics	534
G.13.2.8	Base layer temporal HRD SEI message semantics	535
G.13.2.9	Quality layer integrity check SEI message semantics	536
G.13.2.10	Redundant picture property SEI message semantics	536
G.13.2.11	Temporal level zero dependency representation index SEI message semantics	537
G.13.2.12	Temporal level switching point SEI message semantics	538
G.14	<i>SVC video usability information extension</i>	539
G.14.1	SVC VUI parameters extension syntax	539
G.14.2	SVC VUI parameters extension semantics	539

LIST OF FIGURES

Figure 6-1	– Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame	20
Figure 6-2	– Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields	21
Figure 6-3	– Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a frame	21
Figure 6-4	– Nominal vertical and horizontal sampling locations of 4:2:2 samples top and bottom fields	22
Figure 6-5	– Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a frame	22
Figure 6-6	– Nominal vertical and horizontal sampling locations of 4:4:4 samples top and bottom fields	23
Figure 6-7	– A picture with 11 by 9 macroblocks that is partitioned into two slices	24
Figure 6-8	– Partitioning of the decoded frame into macroblock pairs	24
Figure 6-9	– Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans	26
Figure 6-10	– Scan for 4x4 luma blocks	27
Figure 6-11	– Scan for 8x8 luma blocks	27
Figure 6-12	– Neighbouring macroblocks for a given macroblock	28
Figure 6-13	– Neighbouring macroblocks for a given macroblock in MBAFF frames	29
Figure 6-14	– Determination of the neighbouring macroblock, blocks, and partitions (informative)	30
Figure 7-1	– Structure of an access unit not containing any NAL units with nal_unit_type equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive	64
Figure 8-1	– Intra_4x4 prediction mode directions (informative)	124
Figure 8-2	– Example for temporal direct-mode motion vector inference (informative)	155
Figure 8-3	– Directional segmentation prediction (informative)	156
Figure 8-4	– Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for quarter sample luma interpolation	162
Figure 8-5	– Fractional sample position dependent variables in chroma interpolation and surrounding integer position samples A, B, C, and D	164
Figure 8-6	– Assignment of the indices of dcY to luma4x4BlkIdx	170
Figure 8-7	– Assignment of the indices of dcC to chroma4x4BlkIdx: (a) ChromaArrayType equal to 1, (b) ChromaArrayType equal to 2	172
Figure 8-8	– 4x4 block scans. (a) Zig-zag scan. (b) Field scan (informative)	174
Figure 8-9	– 8x8 block scans. (a) 8x8 zig-zag scan. (b) 8x8 field scan (informative)	175

Figure 8-10 – Boundaries in a macroblock to be filtered.....	194
Figure 8-11 – Convention for describing samples across a 4x4 block horizontal or vertical boundary	198
Figure 9-1 – Illustration of CABAC parsing process for a syntax element SE (informative)	219
Figure 9-2 – Overview of the arithmetic decoding process for a single bin (informative).....	265
Figure 9-3 – Flowchart for decoding a decision	266
Figure 9-4 – Flowchart of renormalization.....	268
Figure 9-5 – Flowchart of bypass decoding process.....	269
Figure 9-6 – Flowchart of decoding a decision before termination	270
Figure 9-7 – Flowchart for encoding a decision	271
Figure 9-8 – Flowchart of renormalization in the encoder	272
Figure 9-9 – Flowchart of PutBit(B).....	272
Figure 9-10 – Flowchart of encoding bypass.....	273
Figure 9-11 – Flowchart of encoding a decision before termination	274
Figure 9-12 – Flowchart of flushing at termination.....	274
Figure C-1 – Structure of byte streams and NAL unit streams for HRD conformance checks	293
Figure C-2 – HRD buffer model.....	294
Figure E-1 – Location of chroma samples for top and bottom fields as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field	353

LIST OF TABLES

Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag... 19	19
Table 6-2 – Specification of input and output assignments for subclauses 6.4.10.1 to 6.4.10.7.....	30
Table 6-3 – Specification of mbAddrN	34
Table 6-4 – Specification of mbAddrN and yM	36
Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes.....	59
Table 7-2 – Assignment of mnemonic names to scaling list indices and specification of fall-back rule.....	69
Table 7-3 – Specification of default scaling lists Default_4x4_Intra and Default_4x4_Inter.....	69
Table 7-4 – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter.....	70
Table 7-5 – Meaning of primary_pic_type	77
Table 7-6 – Name association to slice_type	80
Table 7-7 – reordering_of_pic_nums_idc operations for reordering of reference picture lists.....	85
Table 7-8 – Interpretation of adaptive_ref_pic_marking_mode_flag	87
Table 7-9 – Memory management control operation (memory_management_control_operation) values	88
Table 7-10 – Allowed collective macroblock types for slice_type.....	90
Table 7-11 – Macroblock types for I slices	91
Table 7-12 – Macroblock type with value 0 for SI slices	92
Table 7-13 – Macroblock type values 0 to 4 for P and SP slices.....	93
Table 7-14 – Macroblock type values 0 to 22 for B slices.....	94
Table 7-15 – Specification of CodedBlockPatternChroma values.....	96
Table 7-16 – Relationship between intra_chroma_pred_mode and spatial prediction modes.....	97

Table 7-17 – Sub-macroblock types in P macroblocks.....	98
Table 7-18 – Sub-macroblock types in B macroblocks	99
Table 8-1 – Refined slice group map type.....	108
Table 8-2 – Specification of Intra4x4PredMode[luma4x4BlkIdx] and associated names.....	123
Table 8-3 – Specification of Intra8x8PredMode[luma8x8BlkIdx] and associated names.....	130
Table 8-4 – Specification of Intra16x16PredMode and associated names	137
Table 8-5 – Specification of Intra chroma prediction modes and associated names.....	140
Table 8-6 – Specification of the variable colPic.....	148
Table 8-7 – Specification of PicCodingStruct(X).....	148
Table 8-8 – Specification of mbAddrCol, yM, and vertMvScale	150
Table 8-9 – Assignment of prediction utilization flags.....	152
Table 8-10 – Derivation of the vertical component of the chroma vector in field coding mode	158
Table 8-11 – Differential full-sample luma locations.....	163
Table 8-12 – Assignment of the luma prediction sample predPartLX _L [x _L , y _L]	164
Table 8-13 – Specification of mapping of idx to c _{ij} for zig-zag and field scan.....	174
Table 8-14 – Specification of mapping of idx to c _{ij} for 8x8 zig-zag and 8x8 field scan.....	175
Table 8-15 – Specification of QP _C as a function of qP _L	176
Table 8-16 – Derivation of offset dependent threshold variables α' and β' from indexA and indexB.....	202
Table 8-17 – Value of variable t' _{C0} as a function of indexA and bS	204
Table 9-1 – Bit strings with “prefix” and “suffix” bits and assignment to codeNum ranges (informative).....	206
Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative).....	206
Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v).....	207
Table 9-4 – Assignment of codeNum to values of coded_block_pattern for macroblock prediction modes	207
Table 9-5 – coeff_token mapping to TotalCoeff(coeff_token) and TrailingOnes(coeff_token).....	212
Table 9-6 – Codeword table for level_prefix (informative).....	215
Table 9-7 – total_zeros tables for 4x4 blocks with tzVlcIndex 1 to 7	216
Table 9-8 – total_zeros tables for 4x4 blocks with tzVlcIndex 8 to 15	217
Table 9-9 – total_zeros tables for chroma DC 2x2 and 2x4 blocks.....	217
Table 9-10 – Tables for run_before	218
Table 9-11 – Association of ctxIdx and syntax elements for each slice type in the initialisation process.....	220
Table 9-12 – Values of variables m and n for ctxIdx from 0 to 10.....	221
Table 9-13 – Values of variables m and n for ctxIdx from 11 to 23.....	222
Table 9-14 – Values of variables m and n for ctxIdx from 24 to 39.....	222
Table 9-15 – Values of variables m and n for ctxIdx from 40 to 53.....	222
Table 9-16 – Values of variables m and n for ctxIdx from 54 to 59, and 399 to 401	223
Table 9-17 – Values of variables m and n for ctxIdx from 60 to 69.....	223
Table 9-18 – Values of variables m and n for ctxIdx from 70 to 104.....	224
Table 9-19 – Values of variables m and n for ctxIdx from 105 to 165.....	225
Table 9-20 – Values of variables m and n for ctxIdx from 166 to 226.....	226
Table 9-21 – Values of variables m and n for ctxIdx from 227 to 275.....	227

Table 9-22 – Values of variables m and n for ctxIdx from 277 to 337	228
Table 9-23 – Values of variables m and n for ctxIdx from 338 to 398	229
Table 9-24 – Values of variables m and n for ctxIdx from 402 to 459	230
Table 9-25 – Values of variables m and n for ctxIdx from 460 to 483	231
Table 9-26 – Values of variables m and n for ctxIdx from 484 to 571	231
Table 9-27 – Values of variables m and n for ctxIdx from 572 to 659	233
Table 9-28– Values of variables m and n for ctxIdx from 660 to 717	235
Table 9-29– Values of variables m and n for ctxIdx from 718 to 775	236
Table 9-30 – Values of variables m and n for ctxIdx from 776 to 863	237
Table 9-31 – Values of variables m and n for ctxIdx from 864 to 951	239
Table 9-32 – Values of variables m and n for ctxIdx from 952 to 1011	241
Table 9-33 – Values of variables m and n for ctxIdx from 1012 to 1023	242
Table 9-34 – Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset	243
Table 9-35 – Bin string of the unary binarization (informative)	246
Table 9-36 – Binarization for macroblock types in I slices	248
Table 9-37 – Binarization for macroblock types in P, SP, and B slices	249
Table 9-38 – Binarization for sub-macroblock types in P, SP, and B slices	250
Table 9-39 – Assignment of ctxIdxInc to binIdx for all ctxIdxOffset values except those related to the syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1	252
Table 9-40 – Assignment of ctxIdxBlockCatOffset to ctxBlockCat for syntax elements coded_block_flag, significant_coeff_flag, last_significant_coeff_flag, and coeff_abs_level_minus1	253
Table 9-41 – Specification of ctxIdxInc for specific values of ctxIdxOffset and binIdx	261
Table 9-42 – Specification of ctxBlockCat for the different blocks	262
Table 9-43 – Mapping of scanning position to ctxIdxInc for ctxBlockCat == 5, 9, or 13	263
Table 9-44 – Specification of rangeTabLPS depending on pStateIdx and qCodIRangeIdx	267
Table 9-45 – State transition table	268
Table A-1 – Level limits	283
Table A-2 – Specification of cpbBrVclFactor and cpbBrNalFactor	286
Table A-3 – Baseline profile level limits	286
Table A-4 – Main, High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra profile level limits	287
Table A-5 – Extended profile level limits	288
Table A-6 – Maximum frame rates (frames per second) for some example frame sizes	288
Table D-1 – Interpretation of pic_struct	317
Table D-2 – Mapping of ct_type to source picture scan	318
Table D-3 – Definition of counting_type values	318
Table D-4 – scene_transition_type values	325
Table D-5 – model_id values	332
Table D-6 – blending_mode_id values	334
Table D-7 – filter_hint_type values	340
Table E-1 – Meaning of sample aspect ratio indicator	346

Table E-2 – Meaning of video_format.....	347
Table E-3 – Colour primaries	348
Table E-4 – Transfer characteristics	349
Table E-5 – Matrix coefficients	352
Table E-6 – Divisor for computation of $\Delta t_{n,dpb}(n)$	354
Table G-1 – Name association to slice_type for NAL units with nal_unit_type equal to 20.....	393
Table G-2 – Memory management control operation (memory_management_control_operation) values	401
Table G-3 – Allowed collective macroblock types for slice_type.	403
Table G-4 – Macroblock types for EI slices.	403
Table G-5 – Scale values cS for transform coefficient level scaling	458
Table G-6 – Macroblock type predictors mbTypeILPred.....	475
Table G-7 – Sub-macroblock type predictors subMbTypeILPred[mbPartIdx]	475
Table G-8 – 16-phase luma interpolation filter for resampling in Intra_Base prediction	483
Table G-9 – Mapping of (nX, nY) to coeffTokenIdx and vice versa	505
Table G-10 - Association of ctxIdx and syntax elements for each slice type in the initialisation process.....	508
Table G-11 - Values of variables m and n for ctxIdx from 1024 to 1026.....	508
Table G-12 - Values of variables m and n for ctxIdx from 1027 to 1029	508
Table G-13 - Syntax elements and associated types of binarization, maxBinIdxCtx, and ctxIdxOffset	509
Table G-14 - Assignment of ctxIdxInc to binIdx for the ctxIdxOffset values related to the syntax elements base_mode_flag and residual_prediction_flag.....	510
Table G-15 – Scalable Baseline profile level limits	516
Table G-16 – Specification of cpbBrVclFactor and cpbBrNalFactor.....	517

Foreword

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardising telecommunications on a world-wide basis. The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups that, in turn, produce Recommendations on these topics. The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1. In some areas of information technology that fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialised system for world-wide standardisation. National Bodies that are members of ISO and IEC participate in the development of International Standards through technical committees established by the respective organisation to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organisations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

This Recommendation | International Standard was prepared jointly by ITU-T SG 16 Q.6, also known as VCEG (Video Coding Experts Group), and by ISO/IEC JTC 1/SC 29/WG 11, also known as MPEG (Moving Picture Experts Group). VCEG was formed in 1997 to maintain prior ITU-T video coding standards and develop new video coding standard(s) appropriate for a wide range of conversational and non-conversational services. MPEG was formed in 1988 to establish standards for coding of moving pictures and associated audio for various applications such as digital storage media, distribution, and communication.

In this Recommendation | International Standard Annexes A through E and G contain normative requirements and are an integral part of this Recommendation | International Standard.

ITU-T Recommendation H.264

Advanced video coding for generic audiovisual services

0 Introduction

This clause does not form an integral part of this Recommendation | International Standard.

0.1 Prologue

This subclause does not form an integral part of this Recommendation | International Standard.

As the costs for both processing power and memory have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments. Toward these ends the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) formed a Joint Video Team (JVT) in 2001 for development of a new Recommendation | International Standard.

0.2 Purpose

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard was developed in response to the growing need for higher compression of moving pictures for various applications such as videoconferencing, digital storage media, television broadcasting, internet streaming, and communication. It is also designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments. The use of this Recommendation | International Standard allows motion video to be manipulated as a form of computer data and to be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

0.3 Applications

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard is designed to cover a broad range of applications for video content including but not limited to the following:

CATV	Cable TV on optical networks, copper, etc.
DBS	Direct broadcast satellite video services
DSL	Digital subscriber line video services
DTTB	Digital terrestrial television broadcasting
ISM	Interactive storage media (optical disks, etc.)
MMM	Multimedia mailing
MSPN	Multimedia services over packet networks
RTC	Real-time conversational services (videoconferencing, videophone, etc.)
RVS	Remote video surveillance
SSM	Serial storage media (digital VTR, etc.)

0.4 Publication and versions of this specification

This subclause does not form an integral part of this Recommendation | International Standard.

This specification has been jointly developed by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group. It is published as technically-aligned twin text in both organizations ITU-T and ISO/IEC.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 1 refers to the first approved version of this Recommendation | International Standard. Version 1 was approved by ITU-T on 30 May 2003. The first published version in ISO/IEC corresponded to version 1.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 2 refers to the integrated text containing the corrections specified in the first technical corrigendum. The first fully-published version in the ITU-T was version 2 as approved by ITU-T on 7 May 2004, due to the development of the corrigendum during the publication process. Version 2 was also published in integrated form by ISO/IEC.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 3 refers to the integrated text containing both the first technical corrigendum (2004) and the first amendment, which is referred to as the "Fidelity range extensions". Version 3 was approved by ITU-T on 1 March 2005.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 4 refers to the integrated text containing the first technical corrigendum (2004), the first amendment (the "Fidelity range extensions"), and an additional technical corrigendum (2005). Version 4 was approved by ITU-T on 13 September 2005. In both ITU-T and ISO/IEC, the next complete published version after version 2 was version 4.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 5 refers to the integrated version 4 text with its specification of the High 4:4:4 profile removed.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 6 refers to the integrated version 5 text after its amendment to support additional colour space indicators. In the ITU-T, the changes for versions 5 and 6 were approved on 13 June 2006 and were published as a single amendment.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 7 refers to the integrated version 6 text after its amendment to define five new profiles intended primarily for professional applications (the High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles) and two new types of supplemental enhancement information (SEI) messages (the post-filter hint SEI message and the tone mapping information SEI message). Version 7 was approved by ITU-T on 6 April 2007.

ITU-T Rec. H.264 | ISO/IEC 14496-10 version 8 (the current specification) refers to the integrated version 7 text after its amendment to specify scalable video coding in three profiles (Scalable Baseline, Scalable High, and Scalable High Intra profiles). Version 8 was approved by the ITU-T on 22 November 2007.

0.5 Profiles and levels

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard is designed to be generic in the sense that it serves a wide range of applications, bit rates, resolutions, qualities, and services. Applications should cover, among other things, digital storage media, television broadcasting and real-time communications. In the course of creating this Specification, various requirements from typical applications have been considered, necessary algorithmic elements have been developed, and these have been integrated into a single syntax. Hence, this Specification will facilitate video data interchange among different applications.

Considering the practicality of implementing the full syntax of this Specification, however, a limited number of subsets of the syntax are also stipulated by means of "profiles" and "levels". These and other related terms are formally defined in clause 3.

A "profile" is a subset of the entire bitstream syntax that is specified by this Recommendation | International Standard. Within the bounds imposed by the syntax of a given profile it is still possible to require a very large variation in the performance of encoders and decoders depending upon the values taken by syntax elements in the bitstream such as the specified size of the decoded pictures. In many applications, it is currently neither practical nor economic to implement a decoder capable of dealing with all hypothetical uses of the syntax within a particular profile.

In order to deal with this problem, "levels" are specified within each profile. A level is a specified set of constraints imposed on values of the syntax elements in the bitstream. These constraints may be simple limits on values. Alternatively they may take the form of constraints on arithmetic combinations of values (e.g., picture width multiplied by picture height multiplied by number of pictures decoded per second).

Coded video content conforming to this Recommendation | International Standard uses a common syntax. In order to achieve a subset of the complete syntax, flags, parameters, and other syntax elements are included in the bitstream that signal the presence or absence of syntactic elements that occur later in the bitstream.

0.6 Overview of the design characteristics

This subclause does not form an integral part of this Recommendation | International Standard.

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image quality. With the exception of the transform bypass mode of operation for lossless coding in the High 4:4:4 Intra, CAVLC 4:4:4 Intra, and High 4:4:4 Predictive profiles, and the I_PCM mode of operation in all profiles, the algorithm is typically not lossless, as the exact source sample values are typically not preserved through the encoding and decoding processes. A number of techniques may be used to achieve highly efficient compression. Encoding algorithms (not specified in this Recommendation | International Standard) may select between inter and intra coding for block-shaped regions of each picture. Inter coding uses motion vectors for block-based inter prediction to exploit temporal statistical dependencies between different pictures. Intra coding uses various spatial prediction modes to exploit spatial statistical dependencies in the source signal for a single picture. Motion vectors and intra prediction modes may be specified for a variety of block sizes in the picture. The prediction residual is then further compressed using a transform to remove spatial correlation inside the transform block before it is quantised, producing an irreversible process that typically discards less important visual information while forming a close approximation to the source samples. Finally, the motion vectors or intra prediction modes are combined with the quantised transform coefficient information and encoded using either variable length codes or arithmetic coding.

Scalable video coding is specified in Annex G of this specification allowing the construction of bitstreams that contain sub-bitstreams that conform to this specification. For temporal bitstream scalability, i.e., the presence of a sub-bitstream with a smaller temporal sampling rate than the bitstream, complete access units are removed from the bitstream when deriving the sub-bitstream. In this case, high-level syntax and inter prediction reference pictures in the bitstream are constructed accordingly. For spatial and quality bitstream scalability, i.e. the presence of a sub-bitstream with lower spatial resolution or quality than the bitstream, NAL units are removed from the bitstream when deriving the sub-bitstream. In this case, inter-layer prediction, i.e., the prediction of the higher spatial resolution or quality signal by data of the lower spatial resolution or quality signal, is typically used for efficient coding.

Otherwise, the coding algorithm as described in the previous paragraph is used.

0.6.1 Predictive coding

This subclause does not form an integral part of this Recommendation | International Standard.

Because of the conflicting requirements of random access and highly efficient compression, two main coding types are specified. Intra coding is done without reference to other pictures. Intra coding may provide access points to the coded sequence where decoding can begin and continue correctly, but typically also shows only moderate compression efficiency. Inter coding (predictive or bi-predictive) is more efficient using inter prediction of each block of sample values from some previously decoded picture selected by the encoder. In contrast to some other video coding standards, pictures coded using bi-predictive inter prediction may also be used as references for inter coding of other pictures.

The application of the three coding types to pictures in a sequence is flexible, and the order of the decoding process is generally not the same as the order of the source picture capture process in the encoder or the output order from the decoder for display. The choice is left to the encoder and will depend on the requirements of the application. The decoding order is specified such that the decoding of pictures that use inter-picture prediction follows later in decoding order than other pictures that are referenced in the decoding process.

0.6.2 Coding of progressive and interlaced video

This subclause does not form an integral part of this Recommendation | International Standard.

This Recommendation | International Standard specifies a syntax and decoding process for video that originated in either progressive-scan or interlaced-scan form, which may be mixed together in the same sequence. The two fields of an interlaced frame are separated in capture time while the two fields of a progressive frame share the same capture time. Each field may be coded separately or the two fields may be coded together as a frame. Progressive frames are typically coded as a frame. For interlaced video, the encoder can choose between frame coding and field coding. Frame coding or field coding can be adaptively selected on a picture-by-picture basis and also on a more localized basis within a coded frame. Frame coding is typically preferred when the video scene contains significant detail with limited motion. Field coding typically works better when there is fast picture-to-picture motion.

0.6.3 Picture partitioning into macroblocks and smaller partitions

This subclause does not form an integral part of this Recommendation | International Standard.

As in previous video coding Recommendations and International Standards, a macroblock, consisting of a 16x16 block of luma samples and two corresponding blocks of chroma samples, is used as the basic processing unit of the video decoding process.

A macroblock can be further partitioned for inter prediction. The selection of the size of inter prediction partitions is a result of a trade-off between the coding gain provided by using motion compensation with smaller blocks and the quantity of data needed to represent the data for motion compensation. In this Recommendation | International Standard the inter prediction process can form segmentations for motion representation as small as 4x4 luma samples in size, using motion vector accuracy of one-quarter of the luma sample grid spacing displacement. The process for inter prediction of a sample block can also involve the selection of the picture to be used as the reference picture from a number of stored previously-decoded pictures. Motion vectors are encoded differentially with respect to predicted values formed from nearby encoded motion vectors.

Typically, the encoder calculates appropriate motion vectors and other data elements represented in the video data stream. This motion estimation process in the encoder and the selection of whether to use inter prediction for the representation of each region of the video content is not specified in this Recommendation | International Standard.

0.6.4 Spatial redundancy reduction

This subclause does not form an integral part of this Recommendation | International Standard.

Both source pictures and prediction residuals have high spatial redundancy. This Recommendation | International Standard is based on the use of a block-based transform method for spatial redundancy removal. After inter prediction from previously-decoded samples in other pictures or spatial-based prediction from previously-decoded samples within the current picture, the resulting prediction residual is split into 4x4 blocks. These are converted into the transform domain where they are quantised. After quantisation many of the transform coefficients are zero or have low amplitude and can thus be represented with a small amount of encoded data. The processes of transformation and quantisation in the encoder are not specified in this Recommendation | International Standard.

0.7 How to read this specification

This subclause does not form an integral part of this Recommendation | International Standard.

It is suggested that the reader starts with clause 1 (Scope) and moves on to clause 3 (Definitions). Clause 6 should be read for the geometrical relationship of the source, input, and output of the decoder. Clause 7 (Syntax and semantics) specifies the order to parse syntax elements from the bitstream. See subclauses 7.1-7.3 for syntactical order and see subclause 7.4 for semantics; i.e., the scope, restrictions, and conditions that are imposed on the syntax elements. The actual parsing for most syntax elements is specified in clause 9 (Parsing process). Finally, clause 8 (Decoding process) specifies how the syntax elements are mapped into decoded samples. Throughout reading this specification, the reader should refer to clauses 2 (Normative references), 4 (Abbreviations), and 5 (Conventions) as needed. Annexes A through E also form an integral part of this Recommendation | International Standard.

Annex A specifies eleven profiles (Baseline, Main, Extended, High, High 10, High 4:2:2, High 4:4:4 Predictive, High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, and CAVLC 4:4:4 Intra), each being tailored to certain application domains, and defines the so-called levels of the profiles. Annex B specifies syntax and semantics of a byte stream format for delivery of coded video as an ordered stream of bytes. Annex C specifies the hypothetical reference decoder and its use to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information message payloads. Finally, Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set.

Annex G specifies scalable video coding (SVC). The reader is referred to Annex G for the entire decoding process for SVC, which is specified there with references being made to clauses 2-9 and Annexes A-E. Subclause G.10 specifies three profiles for SVC (Scalable Baseline, Scalable High, and Scalable High Intra).

Throughout this specification, statements appearing with the preamble "NOTE -" are informative and are not an integral part of this Recommendation | International Standard.

1 Scope

This document specifies ITU-T Recommendation H.264 | ISO/IEC International Standard ISO/IEC 14496-10 video coding.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent

edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

- ITU-T Recommendation T.35 (2000), *Procedure for the allocation of ITU-T defined codes for non-standard facilities*.
- ISO/IEC 11578:1996, Annex A, *Universal Unique Identifier*.
- ISO/CIE 10527:1991, *Colorimetric Observers*.

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

- 3.1 **access unit:** A set of *NAL units* always containing exactly one *primary coded picture*. In addition to the *primary coded picture*, an access unit may also contain one or more *redundant coded pictures*, one *auxiliary coded picture*, or other *NAL units* not containing *slices* or *slice data partitions* of a *coded picture*. The decoding of an access unit always results in a *decoded picture*.
- 3.2 **AC transform coefficient:** Any *transform coefficient* for which the *frequency index* in one or both dimensions is non-zero.
- 3.3 **adaptive binary arithmetic decoding process:** An *entropy decoding process* that derives the values of *bins* from a *bitstream* produced by an *adaptive binary arithmetic encoding process*.
- 3.4 **adaptive binary arithmetic encoding process:** An *entropy encoding process*, not normatively specified in this Recommendation | International Standard, that codes a sequence of *bins* and produces a *bitstream* that can be decoded using the *adaptive binary arithmetic decoding process*.
- 3.5 **alpha blending:** A process not specified by this Recommendation | International Standard, in which an *auxiliary coded picture* is used in combination with a *primary coded picture* and with other data not specified by this Recommendation | International Standard in the *display process*. In an alpha blending process, the samples of an *auxiliary coded picture* are interpreted as indications of the degree of opacity (or, equivalently, the degrees of transparency) associated with the corresponding *luma* samples of the *primary coded picture*.
- 3.6 **arbitrary slice order (ASO):** A *decoding order* of *slices* in which the *macroblock address* of the first *macroblock* of some *slice* of a *slice group* may be less than the *macroblock address* of the first *macroblock* of some other preceding *slice* of the same *slice group* or, in the case of a *picture* that is coded using three separate colour planes, some other preceding *slice* of the same *slice group* within the same colour plane, or in which the *slices* of a *slice group* of a *picture* may be interleaved with the *slices* of one or more other *slice groups* of the *picture* or, in the case of a *picture* that is coded using three separate colour planes, with the *slices* of one or more other *slice groups* within the same colour plane.
- 3.7 **auxiliary coded picture:** A *picture* that supplements the *primary coded picture* that may be used in combination with other data not specified by this Recommendation | International Standard in the *display process*. An auxiliary coded picture has the same syntactic and semantic restrictions as a monochrome *redundant coded picture*. An auxiliary coded picture must contain the same number of *macroblocks* as the *primary coded picture*. Auxiliary coded pictures have no normative effect on the *decoding process*. See also *primary coded picture* and *redundant coded picture*.
- 3.8 **B slice:** A *slice* that may be decoded using *intra prediction* or *inter prediction* using at most two *motion vectors* and *reference indices* to *predict* the sample values of each *block*.
- 3.9 **bin:** One bit of a *bin string*.
- 3.10 **binarization:** A set of *bin strings* for all possible values of a *syntax element*.
- 3.11 **binarization process:** A unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*.
- 3.12 **bin string:** A string of *bins*. A bin string is an intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.
- 3.13 **bi-predictive slice:** See *B slice*.
- 3.14 **bitstream:** A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more *coded video sequences*. Bitstream is a collective term used to refer either to a *NAL unit stream* or a *byte stream*.

- 3.15 **block**: An MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*.
- 3.16 **bottom field**: One of two *fields* that comprise a *frame*. Each row of a *bottom field* is spatially located immediately below a corresponding row of a *top field*.
- 3.17 **bottom macroblock (of a macroblock pair)**: The *macroblock* within a *macroblock pair* that contains the samples in the bottom row of samples for the *macroblock pair*. For a *field macroblock pair*, the bottom macroblock represents the samples from the region of the *bottom field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the bottom macroblock represents the samples of the *frame* that lie within the bottom half of the spatial region of the *macroblock pair*.
- 3.18 **broken link**: A location in a *bitstream* at which it is indicated that some subsequent *pictures* in *decoding order* may contain serious visual artefacts due to unspecified operations performed in the generation of the *bitstream*.
- 3.19 **byte**: A sequence of 8 bits, written and read with the most significant bit on the left and the least significant bit on the right. When represented in a sequence of data bits, the most significant bit of a byte is first.
- 3.20 **byte-aligned**: A position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*. A bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned.
- 3.21 **byte stream**: An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.
- 3.22 **can**: A term used to refer to behaviour that is allowed, but not necessarily required.
- 3.23 **category**: A number associated with each *syntax element*. The category is used to specify the allocation of *syntax elements* to *NAL units* for *slice data partitioning*. It may also be used in a manner determined by the application to refer to classes of *syntax elements* in a manner not specified in this Recommendation | International Standard.
- 3.24 **chroma**: An adjective specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours. The symbols used for a chroma array or sample are Cb and Cr.
NOTE – 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 is often associated with the term chrominance.
- 3.25 **coded field**: A *coded representation* of a *field*.
- 3.26 **coded frame**: A *coded representation* of a *frame*.
- 3.27 **coded picture**: A *coded representation* of a *picture*. A coded picture may be either a *coded field* or a *coded frame*. Coded picture is a collective term referring to a *primary coded picture* or a *redundant coded picture*, but not to both together.
- 3.28 **coded picture buffer (CPB)**: A first-in first-out buffer containing *access units* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.
- 3.29 **coded representation**: A data element as represented in its coded form.
- 3.30 **coded slice data partition NAL unit**: A *NAL unit* containing a *slice data partition*.
- 3.31 **coded slice NAL unit**: A *NAL unit* containing a *slice* that is not a *slice* of an *auxiliary coded picture*.
- 3.32 **coded video sequence**: A sequence of *access units* that consists, in decoding order, of an *IDR access unit* followed by zero or more non-IDR *access units* including all subsequent *access units* up to but not including any subsequent *IDR access unit*.
- 3.33 **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that make up a *field* or *frame* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that make up a *field* or *frame* in monochrome format.
- 3.34 **complementary field pair**: A collective term for a *complementary reference field pair* or a *complementary non-reference field pair*.
- 3.35 **complementary non-reference field pair**: Two *non-reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* of opposite parity where the first *field* is not already a paired *field*.

- 3.36 **complementary reference field pair:** Two *reference fields* that are in consecutive *access units* in *decoding order* as two *coded fields* and share the same value of the *frame_num syntax element*, where the second *field* in *decoding order* is not an *IDR picture* and does not include a *memory_management_control_operation syntax element* equal to 5.
- 3.37 **context variable:** A variable specified for the *adaptive binary arithmetic decoding process* of a *bin* by an equation containing recently decoded *bins*.
- 3.38 **DC transform coefficient:** A *transform coefficient* for which the *frequency index* is zero in all dimensions.
- 3.39 **decoded picture:** A *decoded picture* is derived by decoding a *coded picture*. A *decoded picture* is either a decoded *frame*, or a decoded *field*. A decoded *field* is either a decoded *top field* or a decoded *bottom field*.
- 3.40 **decoded picture buffer (DPB):** A buffer holding *decoded pictures* for reference, output reordering, or output delay specified for the *hypothetical reference decoder* in Annex C.
- 3.41 **decoder:** An embodiment of a *decoding process*.
- 3.42 **decoding order:** The order in which *syntax elements* are processed by the *decoding process*.
- 3.43 **decoding process:** The process specified in this Recommendation | International Standard that reads a *bitstream* and derives *decoded pictures* from it.
- 3.44 **direct prediction:** An *inter prediction* for a *block* for which no *motion vector* is decoded. Two direct *prediction* modes are specified that are referred to as *spatial direct prediction* and *temporal prediction* mode.
- 3.45 **display process:** A process not specified in this Recommendation | International Standard having, as its input, the cropped decoded *pictures* that are the output of the *decoding process*.
- 3.46 **decoder under test (DUT):** A *decoder* that is tested for conformance to this Recommendation | International Standard by operating the *hypothetical stream scheduler* to deliver a conforming *bitstream* to the *decoder* and to the *hypothetical reference decoder* and comparing the values and timing of the output of the two *decoders*.
- 3.47 **emulation prevention byte:** A *byte* equal to 0x03 that may be present within a *NAL unit*. The presence of emulation prevention bytes ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* contains a *start code prefix*.
- 3.48 **encoder:** An embodiment of an *encoding process*.
- 3.49 **encoding process:** A process, not specified in this Recommendation | International Standard, that produces a *bitstream* conforming to this Recommendation | International Standard.
- 3.50 **field:** An assembly of alternate rows of a *frame*. A *frame* is composed of two *fields*, a *top field* and a *bottom field*.
- 3.51 **field macroblock:** A *macroblock* containing samples from a single *field*. All *macroblocks* of a *coded field* are field macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be field macroblocks.
- 3.52 **field macroblock pair:** A *macroblock pair* decoded as two *field macroblocks*.
- 3.53 **field scan:** A specific sequential ordering of *transform coefficients* that differs from the *zig-zag scan* by scanning columns more rapidly than rows. Field scan is used for *transform coefficients* in *field macroblocks*.
- 3.54 **flag:** A variable that can take one of the two possible values 0 and 1.
- 3.55 **frame:** A *frame* contains an array of *luma* samples in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format. A *frame* consists of two *fields*, a *top field* and a *bottom field*.
- 3.56 **frame macroblock:** A *macroblock* representing samples from the two *fields* of a *coded frame*. When *macroblock-adaptive frame/field decoding* is not in use, all *macroblocks* of a *coded frame* are frame macroblocks. When *macroblock-adaptive frame/field decoding* is in use, some *macroblocks* of a *coded frame* may be frame macroblocks.
- 3.57 **frame macroblock pair:** A *macroblock pair* decoded as two *frame macroblocks*.
- 3.58 **frequency index:** A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process*.

- 3.59 **hypothetical reference decoder (HRD)**: A hypothetical *decoder* model that specifies constraints on the variability of conforming *NAL unit streams* or conforming *byte streams* that an encoding process may produce.
- 3.60 **hypothetical stream scheduler (HSS)**: A hypothetical delivery mechanism for the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*. The HSS is used for checking the conformance of a *bitstream* or a *decoder*.
- 3.61 **I slice**: A *slice* that is not an *SI slice* that is decoded using *intra prediction* only.
- 3.62 **informative**: A term used to refer to content provided in this Recommendation | International Standard that is not an integral part of this Recommendation | International Standard. Informative content does not establish any mandatory requirements for conformance to this Recommendation | International Standard.
- 3.63 **instantaneous decoding refresh (IDR) access unit**: An *access unit* in which the *primary coded picture* is an *IDR picture*.
- 3.64 **instantaneous decoding refresh (IDR) picture**: A *coded picture* in which all *slices* are *I* or *SI slices* that causes the *decoding process* to mark all *reference pictures* as "unused for reference" immediately after decoding the IDR picture. After the decoding of an IDR picture all following *coded pictures* in *decoding order* can be decoded without *inter prediction* from any *picture* decoded prior to the IDR picture. The first *picture* of each *coded video sequence* is an IDR picture.
- 3.65 **inter coding**: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *inter prediction*.
- 3.66 **inter prediction**: A *prediction* derived from decoded samples of *reference pictures* other than the current decoded *picture*.
- 3.67 **interpretation sample value**: A possibly-altered value corresponding to a decoded sample value of an *auxiliary coded picture* that may be generated for use in the *display process*. Interpretation sample values are not used in the *decoding process* and have no normative effect on the *decoding process*.
- 3.68 **intra coding**: Coding of a *block*, *macroblock*, *slice*, or *picture* that uses *intra prediction*.
- 3.69 **intra prediction**: A *prediction* derived from the decoded samples of the same decoded *slice*.
- 3.70 **intra slice**: See *I slice*.
- 3.71 **inverse transform**: A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values, or by which a set of *transform coefficients* are converted into *DC transform coefficients*.
- 3.72 **layer**: One of a set of syntactical structures in a non-branching hierarchical relationship. Higher layers contain lower layers. The coding layers are the *coded video sequence*, *picture*, *slice*, and *macroblock* layers.
- 3.73 **level**: A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Recommendation | International Standard. The same set of levels is defined for all *profiles*, with most aspects of the definition of each level being in common across different *profiles*. Individual implementations may, within specified constraints, support a different level for each supported *profile*. In a different context, level is the value of a *transform coefficient* prior to *scaling*.
- 3.74 **list 0 (list 1) motion vector**: A *motion vector* associated with a *reference index* pointing into *reference picture list 0 (list 1)*.
- 3.75 **list 0 (list 1) prediction**: *Inter prediction* of the content of a *slice* using a *reference index* pointing into *reference picture list 0 (list 1)*.
- 3.76 **luma**: An adjective specifying that a sample array or single sample is representing the monochrome signal related to the primary colours. The symbol or subscript used for luma is Y or L.
NOTE – 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 is 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.
- 3.77 **macroblock**: A 16x16 *block* of *luma* samples and two corresponding *blocks* of *chroma* samples of a *picture* that has three sample arrays, or a 16x16 *block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes. The division of a *slice* or a *macroblock pair* into macroblocks is a *partitioning*.
- 3.78 **macroblock-adaptive frame/field decoding**: A *decoding process* for *coded frames* in which some *macroblocks* may be decoded as *frame macroblocks* and others may be decoded as *field macroblocks*.

- 3.79 **macroblock address:** When *macroblock-adaptive frame/field decoding* is not in use, a macroblock address is the index of a macroblock in a *macroblock raster scan* of the *picture* starting with zero for the top-left macroblock in a *picture*. When *macroblock-adaptive frame/field decoding* is in use, the macroblock address of the *top macroblock* of a *macroblock pair* is two times the index of the *macroblock pair* in a *macroblock pair raster scan* of the *picture*, and the macroblock address of the *bottom macroblock* of a *macroblock pair* is the macroblock address of the corresponding *top macroblock* plus 1. The macroblock address of the *top macroblock* of each *macroblock pair* is an even number and the macroblock address of the *bottom macroblock* of each *macroblock pair* is an odd number.
- 3.80 **macroblock location:** The two-dimensional coordinates of a *macroblock* in a *picture* denoted by (x, y). For the top left *macroblock* of the *picture* (x, y) is equal to (0, 0). x is incremented by 1 for each *macroblock* column from left to right. When *macroblock-adaptive frame/field decoding* is not in use, y is incremented by 1 for each *macroblock* row from top to bottom. When *macroblock-adaptive frame/field decoding* is in use, y is incremented by 2 for each *macroblock pair* row from top to bottom, and is incremented by an additional 1 when a macroblock is a *bottom macroblock*.
- 3.81 **macroblock pair:** A pair of vertically contiguous *macroblocks* in a *frame* that is coupled for use in *macroblock-adaptive frame/field decoding*. The division of a *slice* into macroblock pairs is a *partitioning*.
- 3.82 **macroblock partition:** A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.
- 3.83 **macroblock to slice group map:** A means of mapping *macroblocks* of a *picture* into *slice groups*. The macroblock to slice group map consists of a list of numbers, one for each coded *macroblock*, specifying the *slice group* to which each coded *macroblock* belongs.
- 3.84 **map unit to slice group map:** A means of mapping *slice group map units* of a *picture* into *slice groups*. The map unit to slice group map consists of a list of numbers, one for each *slice group map unit*, specifying the *slice group* to which each coded *slice group map unit* belongs.
- 3.85 **may:** A term used to refer to behaviour that is allowed, but not necessarily required. In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.
- 3.86 **memory management control operation:** Seven operations that control *reference picture marking*.
- 3.87 **motion vector:** A two-dimensional vector used for *inter prediction* that provides an offset from the coordinates in the *decoded picture* to the coordinates in a *reference picture*.
- 3.88 **must:** A term used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Recommendation | International Standard. This term is used exclusively in an *informative* context.
- 3.89 **NAL unit:** A syntax structure containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.
- 3.90 **NAL unit stream:** A sequence of *NAL units*.
- 3.91 **non-paired field:** A collective term for a *non-paired reference field* or a *non-paired non-reference field*.
- 3.92 **non-paired non-reference field:** A decoded *non-reference field* that is not part of a *complementary non-reference field pair*.
- 3.93 **non-paired reference field:** A decoded *reference field* that is not part of a *complementary reference field pair*.
- 3.94 **non-reference field:** A *field* coded with *nal_ref_idc* equal to 0.
- 3.95 **non-reference frame:** A *frame* coded with *nal_ref_idc* equal to 0.
- 3.96 **non-reference picture:** A *picture* coded with *nal_ref_idc* equal to 0. A *non-reference picture* is not used for *inter prediction* of any other *pictures*.
- 3.97 **note:** A term used to prefix *informative* remarks. This term is used exclusively in an *informative* context.
- 3.98 **opposite parity:** The *opposite parity* of *top* is *bottom*, and vice versa.
- 3.99 **output order:** The order in which the *decoded pictures* are output from the *decoded picture buffer*.

- 3.100 **P slice:** A *slice* that is not an *SP slice* that may be decoded using *intra prediction* or *inter prediction* using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*.
- 3.101 **parameter:** A *syntax element* of a *sequence parameter set* or a *picture parameter set*. Parameter is also used as part of the defined term *quantisation parameter*.
- 3.102 **parity:** The parity of a *field* can be *top* or *bottom*.
- 3.103 **partitioning:** The division of a set into subsets such that each element of the set is in exactly one of the subsets.
- 3.104 **picture:** A collective term for a *field* or a *frame*.
- 3.105 **picture parameter set:** A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by the *pic_parameter_set_id syntax element* found in each *slice header*.
- 3.106 **picture order count:** A variable having a value that is non-decreasing with increasing *picture* position in output order relative to the previous *IDR picture* in *decoding order* or relative to the previous *picture* containing the *memory management control operation* that marks all *reference pictures* as “unused for reference”.
- 3.107 **prediction:** An embodiment of the *prediction process*.
- 3.108 **prediction process:** The use of a *predictor* to provide an estimate of the sample value or data element currently being decoded.
- 3.109 **predictive slice:** See *P slice*.
- 3.110 **predictor:** A combination of specified values or previously decoded sample values or data elements used in the *decoding process* of subsequent sample values or data elements.
- 3.111 **primary coded picture:** The coded representation of a *picture* to be used by the *decoding process* for a bitstream conforming to this Recommendation | International Standard. The primary coded picture contains all *macroblocks* of the *picture*. The only *pictures* that have a normative effect on the *decoding process* are primary coded pictures. See also *redundant coded picture*.
- 3.112 **profile:** A specified subset of the syntax of this Recommendation | International Standard.
- 3.113 **quantisation parameter:** A variable used by the *decoding process* for *scaling* of *transform coefficient levels*.
- 3.114 **random access:** The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.
- 3.115 **raster scan:** A 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.
- 3.116 **raw byte sequence payload (RBSP):** A syntax structure containing an integer number of *bytes* that is encapsulated in a *NAL unit*. An RBSP is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and followed by zero or more subsequent bits equal to 0.
- 3.117 **raw byte sequence payload (RBSP) stop bit:** A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*. The location of the end of the *string of data bits* within an RBSP can be identified by searching from the end of the RBSP for the *RBSP stop bit*, which is the last non-zero bit in the RBSP.
- 3.118 **recovery point:** A point in the *bitstream* at which the recovery of an exact or an approximate representation of the *decoded pictures* represented by the *bitstream* is achieved after a *random access* or *broken link*.
- 3.119 **redundant coded picture:** A coded representation of a *picture* or a part of a *picture*. The content of a redundant coded picture shall not be used by the *decoding process* for a *bitstream* conforming to this Recommendation | International Standard. A *redundant coded picture* is not required to contain all *macroblocks* in the *primary coded picture*. Redundant coded pictures have no normative effect on the *decoding process*. See also *primary coded picture*.
- 3.120 **reference field:** A *reference field* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded field* or *field macroblocks* of a *coded frame* are decoded. See also *reference picture*.
- 3.121 **reference frame:** A *reference frame* may be used for *inter prediction* when *P*, *SP*, and *B slices* of a *coded frame* are decoded. See also *reference picture*.

- 3.122 reference index:** An index into a *reference picture list*.
- 3.123 reference picture:** A *picture* with *nal_ref_idc* not equal to 0. A *reference picture* contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* in *decoding order*.
- 3.124 reference picture list:** A list of *reference pictures* that is used for *inter prediction* of a *P*, *B*, or *SP slice*. For the *decoding process* of a *P* or *SP slice*, there is one reference picture list. For the *decoding process* of a *B slice*, there are two reference picture lists.
- 3.125 reference picture list 0:** A *reference picture list* used for *inter prediction* of a *P*, *B*, or *SP slice*. All *inter prediction* used for *P* and *SP slices* uses reference picture list 0. Reference picture list 0 is one of two *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 1*.
- 3.126 reference picture list 1:** A *reference picture list* used for *inter prediction* of a *B slice*. Reference picture list 1 is one of two lists of *reference picture lists* used for *inter prediction* for a *B slice*, with the other being *reference picture list 0*.
- 3.127 reference picture marking:** Specifies, in the bitstream, how the *decoded pictures* are marked for *inter prediction*.
- 3.128 reserved:** The term reserved, when used in the clauses specifying some values of a particular *syntax element*, are for future use by ITU-T | ISO/IEC. These values shall not be used in *bitstreams* conforming to this Recommendation | International Standard, but may be used in future extensions of this Recommendation | International Standard by ITU-T | ISO/IEC.
- 3.129 residual:** The decoded difference between a *prediction* of a sample or data element and its decoded value.
- 3.130 run:** A number of consecutive data elements represented in the decoding process. In one context, the number of zero-valued *transform coefficient levels* preceding a non-zero *transform coefficient level* in the list of *transform coefficient levels* generated by a *zig-zag scan* or a *field scan*. In other contexts, run refers to a number of *macroblocks*.
- 3.131 sample aspect ratio:** Specifies, for assisting the display process, which is not specified in this Recommendation | International Standard, the ratio between the intended horizontal distance between the columns and the intended vertical distance between the rows of the *luma* sample array in a *frame*. Sample aspect ratio is expressed as $h:v$, where h is horizontal width and v is vertical height (in arbitrary units of spatial distance).
- 3.132 scaling:** The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.
- 3.133 sequence parameter set:** A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded video sequences* as determined by the content of a *seq_parameter_set_id syntax element* found in the *picture parameter set* referred to by the *pic_parameter_set_id syntax element* found in each *slice header*.
- 3.134 shall:** A term used to express mandatory requirements for conformance to this Recommendation | International Standard. When used to express a mandatory constraint on the values of *syntax elements* or on the results obtained by operation of the specified *decoding process*, it is the responsibility of the *encoder* to ensure that the constraint is fulfilled. When used in reference to operations performed by the *decoding process*, any *decoding process* that produces identical results to the *decoding process* described herein conforms to the *decoding process* requirements of this Recommendation | International Standard.
- 3.135 should:** A term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this Recommendation | International Standard.
- 3.136 SI slice:** A *slice* that is coded using *intra prediction* only and using quantisation of the *prediction* samples. An SI slice can be coded such that its decoded samples can be constructed identically to an *SP slice*.
- 3.137 skipped macroblock:** A *macroblock* for which no data is coded other than an indication that the *macroblock* is to be decoded as "skipped". This indication may be common to several *macroblocks*.
- 3.138 slice:** An integer number of *macroblocks* or *macroblock pairs* ordered consecutively in the *raster scan* within a particular *slice group*. For the *primary coded picture*, the division of each *slice group* into slices is a *partitioning*. Although a slice contains *macroblocks* or *macroblock pairs* that are consecutive in the *raster scan* within a *slice group*, these *macroblocks* or *macroblock pairs* are not necessarily consecutive in the *raster scan* within the *picture*. The *macroblock addresses* are derived from the first *macroblock address* in a slice (as represented in the *slice header*) and the *macroblock to slice group map*, and, when a *picture* is coded using three separate colour planes, a colour plane identifier.

- 3.139 **slice data partitioning:** A method of *partitioning* selected *syntax elements* into *syntax structures* based on a *category* associated with each *syntax element*.
- 3.140 **slice group:** A subset of the *macroblocks* or *macroblock pairs* of a *picture*. The division of the *picture* into slice groups is a *partitioning* of the *picture*. The partitioning is specified by the *macroblock to slice group map*.
- 3.141 **slice group map units:** The units of the *map unit to slice group map*.
- 3.142 **slice header:** A part of a coded *slice* containing the data elements pertaining to the first or all *macroblocks* represented in the *slice*.
- 3.143 **source:** Term used to describe the video material or some of its attributes before encoding.
- 3.144 **SP slice:** A *slice* that may be coded using *intra prediction* or *inter prediction* with quantisation of the *prediction* samples using at most one *motion vector* and *reference index* to *predict* the sample values of each *block*. An SP slice can be coded such that its decoded samples can be constructed identically to another SP slice or an *SI slice*.
- 3.145 **start code prefix:** A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*. The location of a start code prefix can be used by a *decoder* to identify the beginning of a new *NAL unit* and the end of a previous *NAL unit*. Emulation of start code prefixes is prevented within *NAL units* by the inclusion of *emulation prevention bytes*.
- 3.146 **string of data bits (SODB):** A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*. Within an SODB, 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.
- 3.147 **sub-macroblock:** One quarter of the samples of a *macroblock*, i.e., an 8x8 *luma block* and two corresponding *chroma blocks* of which one corner is located at a corner of the *macroblock* for a *picture* that has three sample arrays or an 8x8 *luma block* of which one corner is located at a corner of the *macroblock* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.
- 3.148 **sub-macroblock partition:** A *block* of *luma* samples and two corresponding *blocks* of *chroma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction* for a *picture* that has three sample arrays or a *block* of *luma* samples resulting from a *partitioning* of a *sub-macroblock* for *inter prediction* for a monochrome *picture* or a *picture* that is coded using three separate colour planes.
- 3.149 **switching I slice:** See *SI slice*.
- 3.150 **switching P slice:** See *SP slice*.
- 3.151 **syntax element:** An element of data represented in the *bitstream*.
- 3.152 **syntax structure:** Zero or more *syntax elements* present together in the *bitstream* in a specified order.
- 3.153 **top field:** One of two *fields* that comprise a *frame*. Each row of a *top field* is spatially located immediately above the corresponding row of the *bottom field*.
- 3.154 **top macroblock (of a macroblock pair):** The *macroblock* within a *macroblock pair* that contains the samples in the top row of samples for the *macroblock pair*. For a *field macroblock pair*, the top macroblock represents the samples from the region of the *top field* of the *frame* that lie within the spatial region of the *macroblock pair*. For a *frame macroblock pair*, the top macroblock represents the samples of the *frame* that lie within the top half of the spatial region of the *macroblock pair*.
- 3.155 **transform coefficient:** A scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* in an *inverse transform* part of the *decoding process*.
- 3.156 **transform coefficient level:** An integer quantity representing the value associated with a particular two-dimensional frequency index in the *decoding process* prior to *scaling* for computation of a *transform coefficient* value.
- 3.157 **universal unique identifier (UUID):** An identifier that is unique with respect to the space of all universal unique identifiers.
- 3.158 **unspecified:** The term unspecified, when used in the clauses specifying some values of a particular *syntax element*, indicates that the values have no specified meaning in this Recommendation | International Standard and will not have a specified meaning in the future as an integral part of this Recommendation | International Standard.

- 3.159 variable length coding (VLC):** A reversible procedure for entropy coding that assigns shorter bit strings to *symbols* expected to be more frequent and longer bit strings to *symbols* expected to be less frequent.
- 3.160 VCL NAL unit:** A collective term for *coded slice NAL units* and *coded slice data partition NAL units*.
- 3.161 zig-zag scan:** A specific sequential ordering of *transform coefficient levels* from (approximately) the lowest spatial frequency to the highest. Zig-zag scan is used for *transform coefficient levels* in *frame macroblocks*.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply.

CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context-based Adaptive Variable Length Coding
CBR	Constant Bit Rate
CPB	Coded Picture Buffer
DPB	Decoded Picture Buffer
DUT	Decoder Under Test
FIFO	First-In, First-Out
HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
IDR	Instantaneous Decoding Refresh
LSB	Least Significant Bit
MB	Macroblock
MBAFF	Macroblock-Adaptive Frame-Field Coding
MSB	Most Significant Bit
NAL	Network Abstraction Layer
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SODB	String Of Data Bits
SVC	Scalable Video Coding
UUID	Universal Unique Identifier
VBR	Variable Bit Rate
VCL	Video Coding Layer
VLC	Variable Length Coding
VUI	Video Usability Information

5 Conventions

NOTE – The mathematical operators used in this Specification are similar to those used in the C programming language. However, integer division and arithmetic shift operations are specifically defined. Numbering and counting conventions generally begin from 0.

5.1 Arithmetic operators

The following arithmetic operators are defined as follows.

+	Addition
–	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)

*	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 .
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\frac{x}{y}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=x}^y f(i)$	The 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 $y > 0$.

When order of precedence is not indicated explicitly by use of parenthesis, the following rules apply:

- multiplication and division operations are considered to take place before addition and subtraction;
- multiplication and division operations in sequence are evaluated sequentially from left to right;
- addition and subtraction operations in sequence are evaluated sequentially from left to right.

5.2 Logical operators

The following logical operators are defined as follows:

$x \ \&\& \ y$	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

5.3 Relational operators

The following relational operators are defined as follows:

>	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.

5.4 Bit-wise operators

The following bit-wise operators are defined as follows:

&	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.
$x \gg y$	Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for positive integer values of y . Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of x prior to the shift operation.

$x \ll y$ Arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for positive integer values of y . Bits shifted into the LSBs as a result of the left shift have a value equal to 0.

5.5 Assignment operators

The following arithmetic operators are defined as follows:

- = 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 $x = x - (-3)$.

5.6 Range notation

The following notation is used to specify a range of values:

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

5.7 Mathematical functions

The following mathematical functions are defined as follows:

$$\text{Abs}(x) = \begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases} \quad (5-1)$$

$$\text{Ceil}(x) \quad \text{the smallest integer greater than or equal to } x. \quad (5-2)$$

$$\text{Clip1}_Y(x) = \text{Clip3}(0, (1 \ll \text{BitDepth}_Y) - 1, x) \quad (5-3)$$

$$\text{Clip1}_C(x) = \text{Clip3}(0, (1 \ll \text{BitDepth}_C) - 1, x) \quad (5-4)$$

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \quad (5-5)$$

$$\text{Floor}(x) \quad \text{the greatest integer less than or equal to } x. \quad (5-6)$$

$$\text{InverseRasterScan}(a, b, c, d, e) = \begin{cases} (a \% (d/b)) * b; & e = 0 \\ (a / (d/b)) * c; & e = 1 \end{cases} \quad (5-7)$$

$$\text{Log2}(x) \quad \text{returns the base-2 logarithm of } x. \quad (5-8)$$

$$\text{Log10}(x) \quad \text{returns the base-10 logarithm of } x. \quad (5-9)$$

$$\text{Median}(x, y, z) = x + y + z - \text{Min}(x, \text{Min}(y, z)) - \text{Max}(x, \text{Max}(y, z)) \quad (5-10)$$

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases} \quad (5-11)$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases} \quad (5-12)$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \quad (5-13)$$

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x \geq 0 \\ -1 & ; \quad x < 0 \end{cases} \quad (5-14)$$

$$\text{Sqrt}(x) = \sqrt{x} \quad (5-15)$$

5.8 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), its one or two syntax categories, and one or two descriptors 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 letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter may be used in the decoding process for later syntax structures mentioning the originating syntax structure of the variable. Variables 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 are described by their names, which are constructed as syntax element names, 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).

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*}.

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 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 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 other value different than zero.

5.9 Text description of logical operations

In the text, a statement of logical operations as would be described in pseudo-code as

```
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 in pseudo-code as

```
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 any 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 in pseudo-code as

```
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

5.10 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 the 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.

The assignment of variables is specified as follows:

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

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

6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships

6.1 Bitstream formats

This subclause specifies the relationship between the NAL unit stream and byte stream, either of which are referred to as the bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the byte stream 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 byte stream 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 start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes. Methods of framing the NAL units in a manner other than use of the byte stream format are outside the scope of this Recommendation | International Standard. The byte stream format is specified in Annex B.

6.2 Source, decoded, and output picture formats

This subclause specifies the relationship between source and decoded frames and fields that is given via the bitstream.

The video source that is represented by the bitstream is a sequence of either or both frames or fields (called collectively pictures) in decoding order.

The source and decoded pictures (frames or fields) are each comprised of one or more sample arrays:

- Luma (Y) only (monochrome), with or without an auxiliary array.
- Luma and two Chroma (YCbCr or YCgCo), with or without an auxiliary array.
- Green, Blue and Red (GBR, also known as RGB), with or without an auxiliary array.
- Arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ), with or without an auxiliary array.

For convenience of notation and terminology in this specification, 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 (monochrome) auxiliary arrays, which may or may not be present as auxiliary pictures in a coded video sequence, are optional for decoding and can be used for such purposes as alpha blending.

The variables SubWidthC, and SubHeightC are specified in Table 6-1, depending on the chroma format sampling structure, which is specified through `chroma_format_idc` and `separate_colour_plane_flag`. An entry marked as "-" in Table 6-1 denotes an undefined value for SubWidthC or SubHeightC. Other values of `chroma_format_idc`, SubWidthC, and SubHeightC may be specified in the future by ITU-T | ISO/IEC.

Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag

chroma_format_idc	separate_colour_plane_flag	Chroma Format	SubWidthC	SubHeightC
0	0	monochrome	–	–
1	0	4:2:0	2	2
2	0	4:2:2	2	1
3	0	4:4:4	1	1
3	1	4:4:4	-	-

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, depending on the value of separate_colour_plane_flag, the following applies.

- If separate_colour_plane_flag is equal to 0, each of the two chroma arrays has the same height and width as the luma array.
- Otherwise (separate_colour_plane_flag is equal to 1), the three colour planes are separately processed as monochrome sampled pictures.

The width and height of the luma sample arrays are each an integer multiple of 16. In coded video sequences using 4:2:0 chroma sampling, the width and height of chroma sample arrays are each an integer multiple of 8. In coded video sequences using 4:2:2 sampling, the width of the chroma sample arrays is an integer multiple of 8 and the height is an integer multiple of 16. The height of a luma array that is coded as two separate fields or in macroblock-adaptive frame-field coding (see below) is an integer multiple of 32. In coded video sequences using 4:2:0 chroma sampling, the height of each chroma array that is coded as two separate fields or in macroblock-adaptive frame-field coding (see below) is an integer multiple of 16. The width or height of pictures output from the decoding process need not be an integer multiple of 16 and can be specified using a cropping rectangle.

The syntax for the luma and (when present) chroma arrays are ordered such when data for all three colour components is present, the data for the luma array is first, followed by any data for the Cb array, followed by any data for the Cr array, unless otherwise specified.

The width of fields coded referring to a specific sequence parameter set is the same as that of frames coded referring to the same sequence parameter set (see below). The height of fields coded referring to a specific sequence parameter set is half that of frames coded referring to the same sequence parameter set (see below).

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 14, 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 frames are shown in Figure 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).

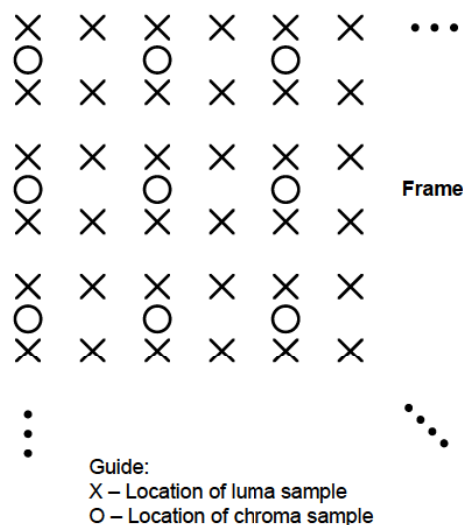


Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a frame

A frame consists of two fields as described below. A coded picture may represent a coded frame or an individual coded field. A coded video sequence conforming to this Recommendation | International Standard may contain arbitrary combinations of coded frames and coded fields. The decoding process is also specified in a manner that allows smaller regions of a coded frame to be coded either as a frame or field region, by use of macroblock-adaptive frame-field coding.

Source and decoded fields are one of two types: top field or bottom field. When two fields are output at the same time, or are combined to be used as a reference frame (see below), the two fields (which shall be of opposite parity) are interleaved. The first (i.e., top), third, fifth, etc., rows of a decoded frame are the top field rows. The second, fourth, sixth, etc., rows of a decoded frame are the bottom field rows. A top field consists of only the top field rows of a decoded frame. When the top field or bottom field of a decoded frame is used as a reference field (see below) only the even rows (for a top field) or the odd rows (for a bottom field) of the decoded frame are used.

When the value of chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in top and bottom fields are shown in Figure 6-2. The nominal vertical sampling relative locations of the chroma samples in a top field are specified as shifted up by one-quarter luma sample height relative to the field-sampling grid. The vertical sampling locations of the chroma samples in a bottom field are specified as shifted down by one-quarter luma sample height relative to the field-sampling grid. Alternative chroma sample relative locations may be indicated in the video usability information (see Annex E).

NOTE – The shifting of the chroma samples is in order for these samples to align vertically to the usual location relative to the full-frame sampling grid as shown in Figure 6-1.

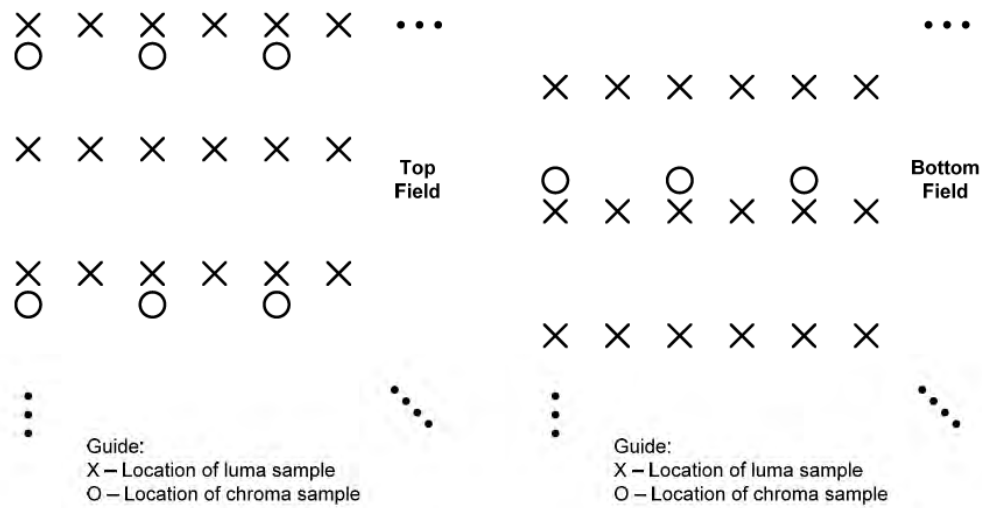


Figure 6-2 – Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields

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 frame and in fields are as shown in Figure 6-3 and Figure 6-4, respectively.

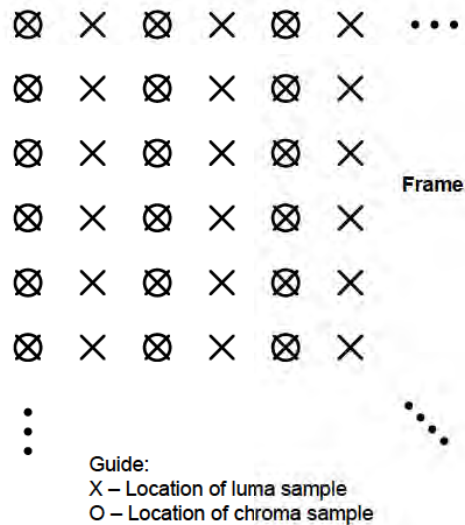


Figure 6-3 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a frame

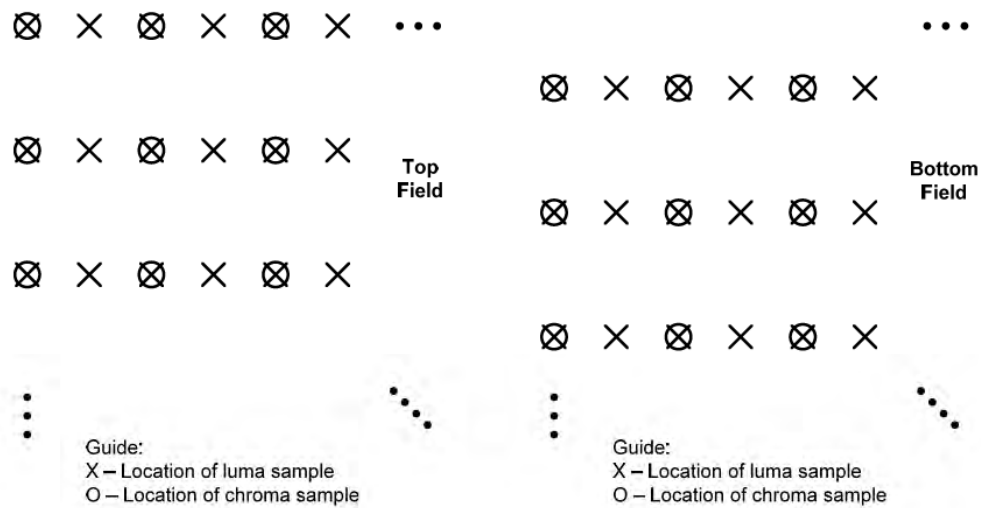


Figure 6-4 – Nominal vertical and horizontal sampling locations of 4:2:2 samples top and bottom fields

When the value of `chroma_format_idc` is equal to 3, all array samples are co-sited for all cases of frames and fields and the nominal locations in a frame and in fields are as shown in Figure 6-5 and 6-6, respectively.

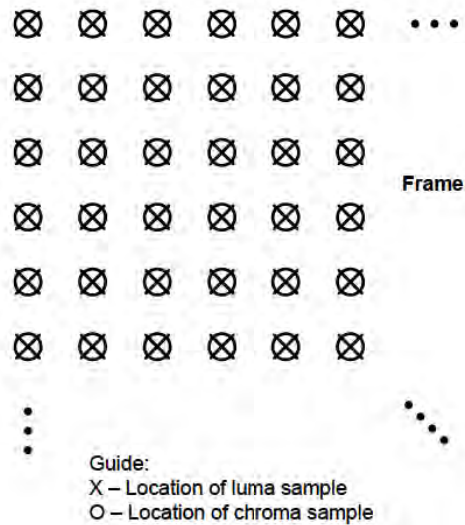


Figure 6-5 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a frame

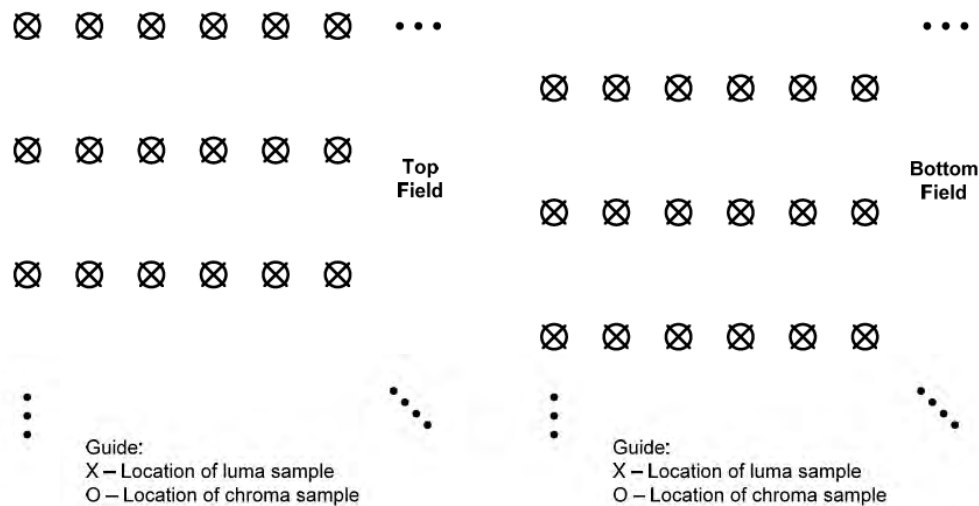


Figure 6-6 – Nominal vertical and horizontal sampling locations of 4:4:4 samples top and bottom fields

The samples are processed in units of macroblocks. The luma array for each macroblock is 16 samples in both width and height. The variables MbWidthC and MbHeightC, which specify the width and height, respectively, of the chroma arrays for each macroblock, are derived as follows.

- If chroma_format_idc is equal to 0 (monochrome) or separate_colour_plane_flag is equal to 1, MbWidthC and MbHeightC are both equal to 0.
- Otherwise, MbWidthC and MbHeightC are derived as

$$\text{MbWidthC} = 16 / \text{SubWidthC} \quad (6-1)$$

$$\text{MbHeightC} = 16 / \text{SubHeightC} \quad (6-2)$$

6.3 Spatial subdivision of pictures and slices

This subclause specifies how a picture is partitioned into slices and macroblocks. Pictures are divided into slices. A slice is a sequence of macroblocks, or, when macroblock-adaptive frame/field decoding is in use, a sequence of macroblock pairs.

Each macroblock is comprised of one 16x16 luma array and, when the chroma sampling format is not equal to 4:0:0 and separate_colour_plane_flag is equal to 0, two corresponding chroma sample arrays. When separate_colour_plane_flag is equal to 1, each macroblock is comprised of one 16x16 luma or chroma sample array. When macroblock-adaptive frame/field decoding is not in use, each macroblock represents a spatial rectangular region of the picture. For example, a picture may be divided into two slices as shown in Figure 6-7.

When a picture is coded using three separate colour planes (separate_colour_plane_flag is equal to 1), a slice contains only macroblocks of one colour component being identified by the corresponding value of colour_plane_id, and each colour component array of a picture consists of slices having the same colour_plane_id value. Coded slices with different values of colour_plane_id within an access unit can be interleaved with each other under the constraint that for each value of colour_plane_id, the coded slice NAL units with that value colour_plane_id shall be in the order of increasing macroblock address for the first macroblock of each coded slice NAL unit.

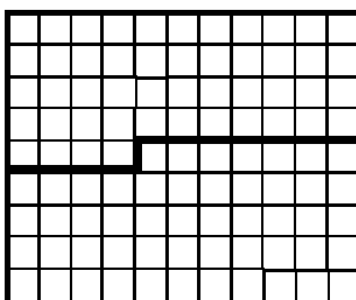


Figure 6-7 – A picture with 11 by 9 macroblocks that is partitioned into two slices

When macroblock-adaptive frame/field decoding is in use, the picture is partitioned into slices containing an integer number of macroblock pairs as shown in Figure 6-8. Each macroblock pair consists of two macroblocks.

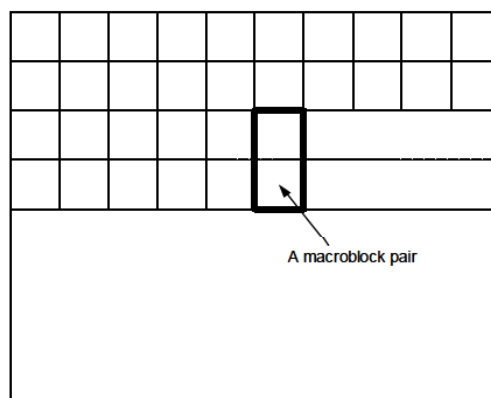


Figure 6-8 – Partitioning of the decoded frame into macroblock pairs

6.4 Inverse scanning processes and derivation processes for neighbours

This subclause specifies inverse scanning processes; i.e., the mapping of indices to locations, and derivation processes for neighbours.

6.4.1 Inverse macroblock scanning process

Input to this process is a macroblock address `mbAddr`.

Output of this process is the location (`x`, `y`) of the upper-left luma sample for the macroblock with address `mbAddr` relative to the upper-left sample of the picture.

The inverse macroblock scanning process is specified as follows.

- If MbaffFrameFlag is equal to 0,

$$x = \text{InverseRasterScan}(\text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 0) \quad (6-3)$$

$$y = \text{InverseRasterScan}(\text{mbAddr}, 16, 16, \text{PicWidthInSamples}_L, 1) \quad (6-4)$$

- Otherwise (MbaffFrameFlag is equal to 1), the following applies.

$$xO = \text{InverseRasterScan}(\text{mbAddr} / 2, 16, 32, \text{PicWidthInSamples}_L, 0) \quad (6-5)$$

$$yO = \text{InverseRasterScan}(\text{mbAddr} / 2, 16, 32, \text{PicWidthInSamples}_L, 1) \quad (6-6)$$

Depending on the current macroblock the following applies.

- If the current macroblock is a frame macroblock

$$x = xO \quad (6-7)$$

$$y = yO + (\text{mbAddr} \% 2) * 16 \quad (6-8)$$

- Otherwise (the current macroblock is a field macroblock),

$$x = xO \quad (6-9)$$

$$y = yO + (\text{mbAddr} \% 2) \quad (6-10)$$

6.4.2 Inverse macroblock partition and sub-macroblock partition scanning process

Macroblocks or sub-macroblocks may be partitioned, and the partitions are scanned for inter prediction as shown in Figure 6-9. The outer rectangles refer to the samples in a macroblock or sub-macroblock, respectively. The rectangles refer to the partitions. The number in each rectangle specifies the index of the inverse macroblock partition scan or inverse sub-macroblock partition scan.

The functions MbPartWidth(), MbPartHeight(), SubMbPartWidth(), and SubMbPartHeight() describing the width and height of macroblock partitions and sub-macroblock partitions are specified in Tables 7-13, 7-14, 7-17, and 7-18. MbPartWidth() and MbPartHeight() are set to appropriate values for each macroblock, depending on the macroblock type. SubMbPartWidth() and SubMbPartHeight() are set to appropriate values for each sub-macroblock of a macroblock with mb_type equal to P_8x8, P_8x8ref0, or B_8x8, depending on the sub-macroblock type.

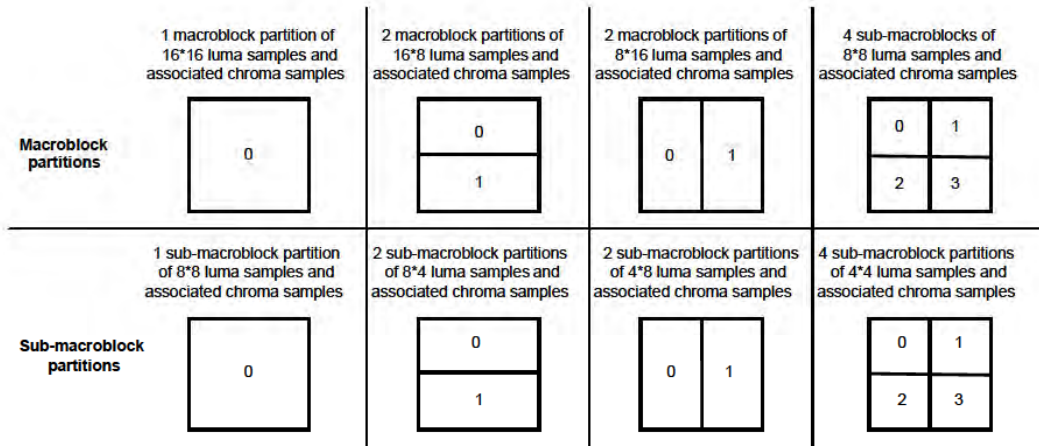


Figure 6-9 – Macroblock partitions, sub-macroblock partitions, macroblock partition scans, and sub-macroblock partition scans

6.4.2.1 Inverse macroblock partition scanning process

Input to this process is the index of a macroblock partition mbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the macroblock partition mbPartIdx relative to the upper-left sample of the macroblock.

The inverse macroblock partition scanning process is specified by

$$x = \text{InverseRasterScan}(\text{mbPartIdx}, \text{MbPartWidth}(\text{mb_type}), \text{MbPartHeight}(\text{mb_type}), 16, 0) \quad (6-11)$$

$$y = \text{InverseRasterScan}(\text{mbPartIdx}, \text{MbPartWidth}(\text{mb_type}), \text{MbPartHeight}(\text{mb_type}), 16, 1) \quad (6-12)$$

6.4.2.2 Inverse sub-macroblock partition scanning process

Inputs to this process are the index of a macroblock partition mbPartIdx and the index of a sub-macroblock partition subMbPartIdx.

Output of this process is the location (x, y) of the upper-left luma sample for the sub-macroblock partition subMbPartIdx relative to the upper-left sample of the sub-macroblock.

The inverse sub-macroblock partition scanning process is specified as follows.

- If mb_type is equal to P_8x8, P_8x8ref0, or B_8x8,

$$x = \text{InverscRasterScan}(\text{subMbPartIdx}, \text{SubMbPartWidth}(\text{sub_mb_type}[\text{mbPartIdx}]), \text{SubMbPartHeight}(\text{sub_mb_type}[\text{mbPartIdx}]), 8, 0) \quad (6-13)$$

$$y = \text{InverseRasterScan}(\text{subMbPartIdx}, \text{SubMbPartWidth}(\text{sub_mb_type}[\text{mbPartIdx}]), \text{SubMbPartHeight}(\text{sub_mb_type}[\text{mbPartIdx}]), 8, 1) \quad (6-14)$$

- Otherwise,

$$x = \text{InverseRasterScan}(\text{subMbPartIdx}, 4, 4, 8, 0) \quad (6-15)$$

$$y = \text{InverseRasterScan}(\text{subMbPartIdx}, 4, 4, 8, 1) \quad (6-16)$$

6.4.3 Inverse 4x4 luma block scanning process

Input to this process is the index of a 4x4 luma block `luma4x4BlkIdx`.

Output of this process is the location (x,y) of the upper-left luma sample for the 4x4 luma block with index `luma4x4BlkIdx` relative to the upper-left luma sample of the macroblock.

Figure 6-10 shows the scan for the 4x4 luma blocks.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 6-10 – Scan for 4x4 luma blocks

The inverse 4x4 luma block scanning process is specified by

$$x = \text{InverseRasterScan}(\text{luma4x4BlkIdx} / 4, 8, 8, 16, 0) + \text{InverseRasterScan}(\text{luma4x4BlkIdx} \% 4, 4, 4, 8, 0) \quad (6-17)$$

$$y = \text{InverseRasterScan}(\text{luma4x4BlkIdx} / 4, 8, 8, 16, 1) + \text{InverseRasterScan}(\text{luma4x4BlkIdx} \% 4, 4, 4, 8, 1) \quad (6-18)$$

6.4.4 Inverse 4x4 Cb or Cr block scanning process for ChromaArrayType equal to 3

This process is only invoked when `ChromaArrayType` is equal to 3.

The inverse 4x4 chroma block scanning process is identical to inverse 4x4 luma block scanning process as specified in subclause 6.4.3 when substituting the term “luma” with the term “Cb” or the term “Cr”, and substituting the term “luma4x4BlkIdx” with the term “cb4x4BlkIdx” or the term “cr4x4BlkIdx” in all places in subclause 6.4.3.

6.4.5 Inverse 8x8 luma block scanning process

Input to this process is the index of an 8x8 luma block `luma8x8BlkIdx`.

Output of this process is the location (x,y) of the upper-left luma sample for the 8x8 luma block with index `luma8x8BlkIdx` relative to the upper-left luma sample of the macroblock.

Figure 6-11 shows the scan for the 8x8 luma blocks.

0	1
2	3

Figure 6-11 – Scan for 8x8 luma blocks

The inverse 8x8 luma block scanning process is specified by

$$x = \text{InverseRasterScan}(\text{luma8x8BlkIdx}, 8, 8, 16, 0) \quad (6-19)$$

$$y = \text{InverseRasterScan}(\text{luma8x8BlkIdx}, 8, 8, 16, 1) \quad (6-20)$$

6.4.6 Inverse 8x8 Cb or Cr block scanning process for ChromaArrayType equal to 3

This process is only invoked when ChromaArrayType is equal to 3.

The inverse 8x8 chroma block scanning process is identical to inverse 8x8 luma block scanning process as specified in subclause 6.4.5 when substituting the term “luma” with the term “Cb” or the term “Cr”, and substituting the term “luma8x8BlkIdx” with the term “cb8x8BlkIdx” or the term “cr8x8BlkIdx” in all places in subclause 6.4.5.

6.4.7 Derivation process of the availability for macroblock addresses

Input to this process is a macroblock address mbAddr.

Output of this process is the availability of the macroblock mbAddr.

NOTE – The meaning of availability is determined when this process is invoked.

The macroblock is marked as available, unless one of the following conditions is true in which case the macroblock is marked as not available:

- mbAddr < 0
- mbAddr > CurrMbAddr
- the macroblock with address mbAddr belongs to a different slice than the macroblock with address CurrMbAddr

6.4.8 Derivation process for neighbouring macroblock addresses and their availability

This process can only be invoked when MbaffFrameFlag is equal to 0.

The outputs of this process are:

- mbAddrA: the address and availability status of the macroblock to the left of the current macroblock.
- mbAddrB: the address and availability status of the macroblock above the current macroblock.
- mbAddrC: the address and availability status of the macroblock above-right of the current macroblock.
- mbAddrD: the address and availability status of the macroblock above-left of the current macroblock.

Figure 6-12 shows the relative spatial locations of the macroblocks with mbAddrA, mbAddrB, mbAddrC, and mbAddrD relative to the current macroblock with CurrMbAddr.

mbAddrD	mbAddrB	mbAddrC
mbAddrA	CurrMbAddr	

Figure 6-12 – Neighbouring macroblocks for a given macroblock

Input to the process in subclause 6.4.7 is $\text{mbAddrA} = \text{CurrMbAddr} - 1$ and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when $\text{CurrMbAddr} \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.4.7 is $\text{mbAddrB} = \text{CurrMbAddr} - \text{PicWidthInMbs}$ and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.7 is $\text{mbAddrC} = \text{CurrMbAddr} - \text{PicWidthInMbs} + 1$ and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when $(\text{CurrMbAddr} + 1) \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.4.7 is $\text{mbAddrD} = \text{CurrMbAddr} - \text{PicWidthInMbs} - 1$ and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when $\text{CurrMbAddr} \% \text{PicWidthInMbs}$ is equal to 0.

6.4.9 Derivation process for neighbouring macroblock addresses and their availability in MBAFF frames

This process can only be invoked when MbaffFrameFlag is equal to 1.

The outputs of this process are

- mbAddrA : the address and availability status of the top macroblock of the macroblock pair to the left of the current macroblock pair.
- mbAddrB : the address and availability status of the top macroblock of the macroblock pair above the current macroblock pair.
- mbAddrC : the address and availability status of the top macroblock of the macroblock pair above-right of the current macroblock pair.
- mbAddrD : the address and availability status of the top macroblock of the macroblock pair above-left of the current macroblock pair.

Figure 6-13 shows the relative spatial locations of the macroblocks with mbAddrA , mbAddrB , mbAddrC , and mbAddrD relative to the current macroblock with CurrMbAddr .

mbAddrA , mbAddrB , mbAddrC , and mbAddrD have identical values regardless whether the current macroblock is the top or the bottom macroblock of a macroblock pair.

mbAddrD	mbAddrB	mbAddrC
mbAddrA	CurrMbAddr or CurrMbAddr	

Figure 6-13 – Neighbouring macroblocks for a given macroblock in MBAFF frames

Input to the process in subclause 6.4.7 is $\text{mbAddrA} = 2 * (\text{CurrMbAddr} / 2 - 1)$ and the output is whether the macroblock mbAddrA is available. In addition, mbAddrA is marked as not available when $(\text{CurrMbAddr} / 2) \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.4.7 is $\text{mbAddrB} = 2 * (\text{CurrMbAddr} / 2 - \text{PicWidthInMbs})$ and the output is whether the macroblock mbAddrB is available.

Input to the process in subclause 6.4.7 is $\text{mbAddrC} = 2 * (\text{CurrMbAddr} / 2 - \text{PicWidthInMbs} + 1)$ and the output is whether the macroblock mbAddrC is available. In addition, mbAddrC is marked as not available when $(\text{CurrMbAddr} / 2 + 1) \% \text{PicWidthInMbs}$ is equal to 0.

Input to the process in subclause 6.4.7 is $\text{mbAddrD} = 2 * (\text{CurrMbAddr} / 2 - \text{PicWidthInMbs} - 1)$ and the output is whether the macroblock mbAddrD is available. In addition, mbAddrD is marked as not available when $(\text{CurrMbAddr} / 2) \% \text{PicWidthInMbs}$ is equal to 0.

6.4.10 Derivation processes for neighbouring macroblocks, blocks, and partitions

Subclause 6.4.10.1 specifies the derivation process for neighbouring macroblocks.

Subclause 6.4.10.2 specifies the derivation process for neighbouring 8x8 luma blocks.

Subclause 6.4.10.3 specifies the derivation process for neighbouring 8x8 chroma blocks for ChromaArrayType equal to 3.

Subclause 6.4.10.4 specifies the derivation process for neighbouring 4x4 luma blocks.

Subclause 6.4.10.5 specifies the derivation process for neighbouring 4x4 chroma blocks.

Subclause 6.4.10.6 specifies the derivation process for neighbouring 4x4 chroma blocks for ChromaArrayType equal to 3.

Subclause 6.4.10.7 specifies the derivation process for neighbouring partitions.

Table 6-2 specifies the values for the difference of luma location (x_D , y_D) for the input and the replacement for N in mbAddrN, mbPartIdxN, subMbPartIdxN, luma8x8BlkIdxN, cb8x8BlkIdxN, cr8x8BlkIdxN, luma4x4BlkIdxN, cb4x4BlkIdxN, cr4x4BlkIdxN, and chroma4x4BlkIdxN for the output. These input and output assignments are used in subclauses 6.4.10.1 to 6.4.10.7. The variable predPartWidth is specified when Table 6-2 is referred to.

Table 6-2 – Specification of input and output assignments for subclauses 6.4.10.1 to 6.4.10.7

N	x_D	y_D
A	-1	0
B	0	-1
C	predPartWidth	-1
D	-1	-1

Figure 6-14 illustrates the relative location of the neighbouring macroblocks, blocks, or partitions A, B, C, and D to the current macroblock, partition, or block, when the current macroblock, partition, or block is in frame coding mode.

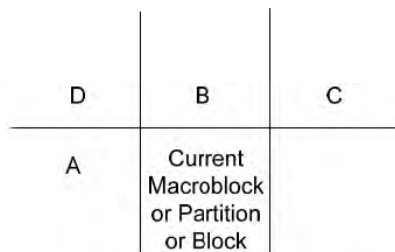


Figure 6-14 – Determination of the neighbouring macroblock, blocks, and partitions (informative)

6.4.10.1 Derivation process for neighbouring macroblocks

Outputs of this process are

- mbAddrA: the address of the macroblock to the left of the current macroblock and its availability status and
 - mbAddrB: the address of the macroblock above the current macroblock and its availability status.
- mbAddrN (with N being A or B) is derived as follows:
- The difference of luma location (x_D , y_D) is set according to Table 6-2.
 - The derivation process for neighbouring locations as specified in subclause 6.4.11 is invoked for luma locations with (x_N , y_N) equal to (x_D , y_D), and the output is assigned to mbAddrN.

6.4.10.2 Derivation process for neighbouring 8x8 luma block

Input to this process is an 8x8 luma block index luma8x8BlkIdx.

The luma8x8BlkIdx specifies the 8x8 luma blocks of a macroblock in a raster scan.

Outputs of this process are

- mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
- luma8x8BlkIdxA: the index of the 8x8 luma block to the left of the 8x8 block with index luma8x8BlkIdx and its availability status,
- mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
- luma8x8BlkIdxB: the index of the 8x8 luma block above the 8x8 block with index luma8x8BlkIdx and its availability status.

mbAddrN and luma8x8BlkIdxN (with N being A or B) are derived as follows.

- The difference of luma location (xD, yD) is set according to Table 6-2.
- The luma location (xN, yN) is specified by

$$xN = (\text{luma8x8BlkIdx} \% 2) * 8 + xD \quad (6-21)$$

$$yN = (\text{luma8x8BlkIdx} / 2) * 8 + yD \quad (6-22)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.11 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
- The variable luma8x8BlkIdxN is derived as follows.
 - If mbAddrN is not available, luma8x8BlkIdxN is marked as not available.
 - Otherwise (mbAddrN is available), the 8x8 luma block in the macroblock mbAddrN covering the luma location (xW, yW) is assigned to luma8x8BlkIdxN.

6.4.10.3 Derivation process for neighbouring 8x8 chroma blocks for ChromaArrayType equal to 3

This process is only invoked when ChromaArrayType is equal to 3.

The derivation process for neighbouring 8x8 chroma block is identical to the derivation process for neighbouring 8x8 luma block as specified in subclause 6.4.10.2 when substituting the term “luma” with the term “Cb” or the term “Cr”, and substituting the term “luma8x8BlkIdx” with the term “cb8x8BlkIdx” or the term “cr8x8BlkIdx” in all places in subclause 6.4.10.2.

6.4.10.4 Derivation process for neighbouring 4x4 luma blocks

Input to this process is a 4x4 luma block index luma4x4BlkIdx.

Outputs of this process are

- mbAddrA: either equal to CurrMbAddr or the address of the macroblock to the left of the current macroblock and its availability status,
- luma4x4BlkIdxA: the index of the 4x4 luma block to the left of the 4x4 block with index luma4x4BlkIdx and its availability status,
- mbAddrB: either equal to CurrMbAddr or the address of the macroblock above the current macroblock and its availability status,
- luma4x4BlkIdxB: the index of the 4x4 luma block above the 4x4 block with index luma4x4BlkIdx and its availability status.

mbAddrN and luma4x4BlkIdxN (with N being A or B) are derived as follows.

- The difference of luma location (xD, yD) is set according to Table 6-2.
- The inverse 4x4 luma block scanning process as specified in subclause 6.4.3 is invoked with luma4x4BlkIdx as the input and (x, y) as the output.
- The luma location (xN, yN) is specified by

$$xN = x + xD \quad (6-23)$$

$$yN = y + yD \quad (6-24)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.11 is invoked for luma locations with (x_N , y_N) as the input and the output is assigned to $mbAddrN$ and (x_W , y_W).
- The variable $luma4x4BlkIdxN$ is derived as follows:
 - If $mbAddrN$ is not available, $luma4x4BlkIdxN$ is marked as not available.
 - Otherwise ($mbAddrN$ is available), the 4x4 luma block in the macroblock $mbAddrN$ covering the luma location (x_W , y_W) is assigned to $luma4x4BlkIdxN$.

6.4.10.5 Derivation process for neighbouring 4x4 chroma blocks

Input to this process is a 4x4 chroma block index $chroma4x4BlkIdx$.

Outputs of this process are

- $mbAddrA$ (either equal to $CurrMbAddr$ or the address of the macroblock to the left of the current macroblock) and its availability status,
- $chroma4x4BlkIdxA$ (the index of the 4x4 chroma block to the left of the 4x4 chroma block with index $chroma4x4BlkIdx$) and its availability status,
- $mbAddrB$ (either equal to $CurrMbAddr$ or the address of the macroblock above the current macroblock) and its availability status,
- $chroma4x4BlkIdxB$ (the index of the 4x4 chroma block above the 4x4 chroma block with index $chroma4x4BlkIdx$) and its availability status.

$mbAddrN$ and $chroma4x4BlkIdxN$ (with N being A or B) are derived as follows:

- The difference of chroma location (x_D , y_D) is set according to Table 6-2.
- The position (x , y) of the upper-left sample of the 4x4 chroma block with index $chroma4x4BlkIdx$ is derived by

$$x = \text{InverseRasterScan}(chroma4x4BlkIdx, 4, 4, 8, 0) \quad (6-25)$$

$$y = \text{InverseRasterScan}(chroma4x4BlkIdx, 4, 4, 8, 1) \quad (6-26)$$

- The chroma location (x_N , y_N) is specified by

$$x_N = x + x_D \quad (6-27)$$

$$y_N = y + y_D \quad (6-28)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.11 is invoked for chroma locations with (x_N , y_N) as the input and the output is assigned to $mbAddrN$ and (x_W , y_W).
- The variable $chroma4x4BlkIdxN$ is derived as follows.
 - If $mbAddrN$ is not available, $chroma4x4BlkIdxN$ is marked as not available.
 - Otherwise ($mbAddrN$ is available), the 4x4 chroma block in the macroblock $mbAddrN$ covering the chroma location (x_W , y_W) is assigned to $chroma4x4BlkIdxN$.

6.4.10.6 Derivation process for neighbouring 4x4 chroma blocks for ChromaArrayType equal to 3

This process is only invoked when $ChromaArrayType$ is equal to 3.

The derivation process for neighbouring 4x4 chroma block in 4:4:4 chroma format is identical to the derivation process for neighbouring 4x4 luma block as specified in subclause 6.4.10.4 when substituting the term “luma” with the term “Cb” or the term “Cr”, and substituting the term “ $luma4x4BlkIdx$ ” with the term “ $cb4x4BlkIdx$ ” or the term “ $cr4x4BlkIdx$ ” in all places in subclause 6.4.10.4.

6.4.10.7 Derivation process for neighbouring partitions

Inputs to this process are

- a macroblock partition index $mbPartIdx$
- a current sub-macroblock type $currSubMbType$
- a sub-macroblock partition index $subMbPartIdx$

Outputs of this process are

- mbAddrA\mbPartIdxA\subMbPartIdxA: specifying the macroblock or sub-macroblock partition to the left of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,
- mbAddrB\mbPartIdxB\subMbPartIdxB: specifying the macroblock or sub-macroblock partition above the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,
- mbAddrC\mbPartIdxC\subMbPartIdxC: specifying the macroblock or sub-macroblock partition to the right-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status,
- mbAddrD\mbPartIdxD\subMbPartIdxD: specifying the macroblock or sub-macroblock partition to the left-above of the current macroblock and its availability status, or the sub-macroblock partition CurrMbAddr\mbPartIdx\subMbPartIdx and its availability status.

mbAddrN, mbPartIdxN, and subMbPartIdxN (with N being A, B, C, or D) are derived as follows.

- The inverse macroblock partition scanning process as described in subclause 6.4.2.1 is invoked with mbPartIdx as the input and (x, y) as the output.
- The location of the upper-left luma sample inside a macroblock partition (xS, yS) is derived as follows.
 - If mb_type is equal to P_8x8, P_8x8ref0 or B_8x8, the inverse sub-macroblock partition scanning process as described in subclause 6.4.2.2 is invoked with subMbPartIdx as the input and (xS, yS) as the output.
 - Otherwise, (xS, yS) are set to (0, 0).
- The variable predPartWidth in Table 6-2 is specified as follows.
 - If mb_type is equal to P_Skip, B_Skip, or B_Direct_16x16, predPartWidth = 16.
 - Otherwise, if mb_type is equal to B_8x8, the following applies.
 - If currSubMbType is equal to B_Direct_8x8, predPartWidth = 16.
NOTE 1 – When currSubMbType is equal to B_Direct_8x8 and direct_spatial_mv_pred_flag is equal to 1, the predicted motion vector is the predicted motion vector for the complete macroblock.
 - Otherwise, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).
 - Otherwise, if mb_type is equal to P_8x8 or P_8x8ref0, predPartWidth = SubMbPartWidth(sub_mb_type[mbPartIdx]).
 - Otherwise, predPartWidth = MbPartWidth(mb_type).
- The difference of luma location (xD, yD) is set according to Table 6-2.
- The neighbouring luma location (xN, yN) is specified by

$$xN = x + xS + xD \quad (6-29)$$

$$yN = y + yS + yD \quad (6-30)$$

- The derivation process for neighbouring locations as specified in subclause 6.4.11 is invoked for luma locations with (xN, yN) as the input and the output is assigned to mbAddrN and (xW, yW).
- Depending on mbAddrN, the following applies.
 - If mbAddrN is not available, the macroblock or sub-macroblock partition mbAddrN\mbPartIdxN\subMbPartIdxN is marked as not available.
 - Otherwise (mbAddrN is available), the following applies.
 - The macroblock partition in the macroblock mbAddrN covering the luma location (xW, yW) is assigned to mbPartIdxN and the sub-macroblock partition inside the macroblock partition mbPartIdxN covering the sample (xW, yW) in the macroblock mbAddrN is assigned to subMbPartIdxN.
 - When the partition given by mbPartIdxN and subMbPartIdxN is not yet decoded, the macroblock partition mbPartIdxN and the sub-macroblock partition subMbPartIdxN are marked as not available.

NOTE 2 – The latter condition is, for example, the case when $mbPartIdx = 2$, $subMbPartIdx = 3$, $xD = 4$, $yD = -1$, i.e., when neighbour C of the last 4x4 luma block of the third sub-macroblock is requested.

6.4.11 Derivation process for neighbouring locations

Input to this process is a luma or chroma location (xN, yN) expressed relative to the upper left corner of the current macroblock.

Outputs of this process are

- $mbAddrN$: either equal to $CurrMbAddr$ or to the address of neighbouring macroblock that contains (xN, yN) and its availability status,
- (xW, yW) : the location (xN, yN) expressed relative to the upper-left corner of the macroblock $mbAddrN$ (rather than relative to the upper-left corner of the current macroblock).

Let $maxW$ and $maxH$ be variables specifying maximum values of the location components xN , xW , and yN , yW , respectively. $maxW$ and $maxH$ are derived as follows:

- If this process is invoked for neighbouring luma locations,

$$maxW = maxH = 16 \quad (6-31)$$

- Otherwise (this process is invoked for neighbouring chroma locations),

$$maxW = MbWidthC \quad (6-32)$$

$$maxH = MbHeightC \quad (6-33)$$

Depending on the variable $MbaffFrameFlag$, the neighbouring locations are derived as follows:

- If $MbaffFrameFlag$ is equal to 0, the specification for neighbouring locations in fields and non-MBAFF frames as described in subclause 6.4.11.1 is applied.
- Otherwise ($MbaffFrameFlag$ is equal to 1), the specification for neighbouring locations in MBAFF frames as described in subclause 6.4.11.2 is applied.

6.4.11.1 Specification for neighbouring locations in fields and non-MBAFF frames

The specifications in this subclause are applied when $MbaffFrameFlag$ is equal to 0.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.8 is invoked with $mbAddrA$, $mbAddrB$, $mbAddrC$, and $mbAddrD$ as well as their availability status as the output.

Table 6-3 specifies $mbAddrN$ depending on (xN, yN) .

Table 6-3 – Specification of $mbAddrN$

xN	yN	$mbAddrN$
< 0	< 0	$mbAddrD$
< 0	$0 \dots maxH - 1$	$mbAddrA$
$0 \dots maxW - 1$	< 0	$mbAddrB$
$0 \dots maxW - 1$	$0 \dots maxH - 1$	$CurrMbAddr$
$> maxW - 1$	< 0	$mbAddrC$
$> maxW - 1$	$0 \dots maxH - 1$	not available
	$> maxH - 1$	not available

The neighbouring location (xW , yW) relative to the upper-left corner of the macroblock $mbAddrN$ is derived as

$$xW = (xN + \max W) \% \max W \quad (6-34)$$

$$yW = (yN + \max H) \% \max H \quad (6-35)$$

6.4.11.2 Specification for neighbouring locations in MBAFF frames

The specifications in this subclause are applied when $MbaffFrameFlag$ is equal to 1.

The derivation process for neighbouring macroblock addresses and their availability in subclause 6.4.9 is invoked with $mbAddrA$, $mbAddrB$, $mbAddrC$, and $mbAddrD$ as well as their availability status as the output.

Table 6-4 specifies the macroblock addresses $mbAddrN$ and yM in two ordered steps:

1. Specification of a macroblock address $mbAddrX$ depending on (xN , yN) and the following variables:
 - The variable $currMbFrameFlag$ is derived as follows.
 - If the macroblock with address $CurrMbAddr$ is a frame macroblock, $currMbFrameFlag$ is set equal to 1,
 - Otherwise (the macroblock with address $CurrMbAddr$ is a field macroblock), $currMbFrameFlag$ is set equal to 0.
 - The variable $mbIsTopMbFlag$ is derived as follows.
 - If the macroblock with address $CurrMbAddr$ is a top macroblock ($CurrMbAddr \% 2$ is equal to 0), $mbIsTopMbFlag$ is set equal to 1;
 - Otherwise (the macroblock with address $CurrMbAddr$ is a bottom macroblock, $CurrMbAddr \% 2$ is equal to 1), $mbIsTopMbFlag$ is set equal to 0.
2. Depending on the availability of $mbAddrX$, the following applies:
 - If $mbAddrX$ is not available, $mbAddrN$ is marked as not available.
 - Otherwise ($mbAddrX$ is available), $mbAddrN$ is marked as available and Table 6-4 specifies $mbAddrN$ and yM depending on (xN , yN), $currMbFrameFlag$, $mbIsTopMbFlag$, and the variable $mbAddrXFrameFlag$, which is derived as follows.
 - If the macroblock $mbAddrX$ is a frame macroblock, $mbAddrXFrameFlag$ is set equal to 1,
 - Otherwise (the macroblock $mbAddrX$ is a field macroblock), $mbAddrXFrameFlag$ is set equal to 0.

Unspecified values (na) of the above flags in Table 6-4 indicate that the value of the corresponding flag is not relevant for the current table rows.

Table 6-4 – Specification of mbAddrN and yM

x_N	y_N	currMbFrameFlag	mbIsTopMbFlag	mbAddrX	mbAddrXFrameFlag	additional condition	mbAddrN	y_M
< 0	< 0	1	1	mbAddrD			mbAddrD + 1	y_N
			0	mbAddrA	1		mbAddrA	y_N
		0			0		mbAddrA + 1	$(y_N + \text{maxH}) \gg 1$
			1	mbAddrD	1		mbAddrD + 1	$2 * y_N$
					0		mbAddrD	y_N
			0	mbAddrD			mbAddrD + 1	y_N
< 0	$0 \dots \text{maxH} - 1$	1	1	mbAddrA	1		mbAddrA	y_N
					0	$y_N \% 2 == 0$	mbAddrA	$y_N \gg 1$
						$y_N \% 2 != 0$	mbAddrA + 1	$y_N \gg 1$
			0	mbAddrA	1		mbAddrA + 1	y_N
		0			0	$y_N \% 2 == 0$	mbAddrA	$(y_N + \text{maxH}) \gg 1$
						$y_N \% 2 != 0$	mbAddrA + 1	$(y_N + \text{maxH}) \gg 1$
			1	mbAddrA	1	$y_N < (\text{maxH} / 2)$	mbAddrA	$y_N \ll 1$
					0	$y_N \geq (\text{maxH} / 2)$	mbAddrA + 1	$(y_N \ll 1) - \text{maxH}$
							mbAddrA	y_N
			0	mbAddrA	1	$y_N < (\text{maxH} / 2)$	mbAddrA	$(y_N \ll 1) + 1$
					0	$y_N \geq (\text{maxH} / 2)$	mbAddrA + 1	$(y_N \ll 1) + 1 - \text{maxH}$
							mbAddrA + 1	y_N
$0 \dots \text{maxW} - 1$	< 0	1	1	mbAddrB			mbAddrB + 1	y_N
			0	CurrMbAddr			CurrMbAddr - 1	y_N
		0	1	mbAddrB	1		mbAddrB + 1	$2 * y_N$
			0	mbAddrB	0		mbAddrB	y_N
$0 \dots \text{maxW} - 1$	$0 \dots \text{maxH} - 1$			CurrMbAddr			CurrMbAddr	y_N
> $\text{maxW} - 1$	< 0	1	1	mbAddrC			mbAddrC + 1	y_N
			0	not available			not available	na
		0	1	mbAddrC	1		mbAddrC + 1	$2 * y_N$
					0		mbAddrC	y_N
> $\text{maxW} - 1$	$0 \dots \text{maxH} - 1$			not available			not available	na
	> $\text{maxH} - 1$			not available			not available	na

The neighbouring luma location (x_W , y_W) relative to the upper-left corner of the macroblock mbAddrN is derived as

$$x_W = (x_N + \text{maxW}) \% \text{maxW} \quad (6-36)$$

$$y_W = (y_M + \text{maxH}) \% \text{maxH} \quad (6-37)$$

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. Additional constraints on the syntax may be specified, either directly or indirectly, in other clauses.

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

The following table lists examples of pseudo code used to describe the syntax. 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.

	C	Descriptor
/* A statement can be a syntax element with an associated syntax category and 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	3	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, categories, and descriptors

The functions presented here are used in the syntactical description. These functions 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.

`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_byte_stream()`, which is used only in the byte stream NAL unit syntax structure specified in Annex B, is specified as follows:

- If more data follow in the byte stream, the return value of `more_data_in_byte_stream()` is equal to TRUE.
- Otherwise, the return value of `more_data_in_byte_stream()` is equal to FALSE.

`more_rbsp_data()` is specified as follows:

- If there is more data in an RBSP before `rbsp_trailing_bits()`, 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 byte stream 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. When used within the byte stream as specified in Annex B, `next_bits(n)` returns a value of 0 if fewer than *n* bits remain within the byte stream.

`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.

Categories (labelled in the table as C) specify the partitioning of slice data into at most three slice data partitions. Slice data partition A contains all syntax elements of category 2. Slice data partition B contains all syntax elements of category 3. Slice data partition C contains all syntax elements of category 4. The meaning of other category values is not specified. For some syntax elements, two category values, separated by a vertical bar, are used. In these cases, the category value to be applied is further specified in the text. For syntax structures used within other syntax structures, the categories of all syntax elements found within the included syntax structure are listed, separated by a vertical bar. A syntax element or syntax structure with category marked as "All" is present within all syntax structures that include that syntax element or syntax structure. For syntax structures used within other syntax structures, a numeric category value provided in a syntax table at the location of the inclusion of a syntax structure containing a syntax element with category marked as "All" is considered to apply to the syntax elements with category "All".

The following descriptors specify the parsing process of each syntax element. For some syntax elements, two descriptors, separated by a vertical bar, are used. In these cases, the left descriptors apply when `entropy_coding_mode_flag` is equal to 0 and the right descriptor applies when `entropy_coding_mode_flag` is equal to 1.

- `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)`.
- `ce(v)`: context-adaptive variable-length entropy-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.
- `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.

- `me(v)`: mapped Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- `se(v)`: signed integer Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- `te(v)`: truncated Exp-Golomb-coded syntax element with left bit first. The parsing process for this descriptor is specified in subclause 9.1.
- `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 Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.1.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

<code>nal_unit(NumBytesInNALunit) {</code>	C	Descriptor
<code> forbidden_zero_bit</code>	All	<code>f(1)</code>
<code> nal_ref_idc</code>	All	<code>u(2)</code>
<code> nal_unit_type</code>	All	<code>u(5)</code>
<code> NumBytesInRBSP = 0</code>		
<code> nalUnitHeaderBytes = 1</code>		
<code> if(nal_unit_type == 14 nal_unit_type == 20) {</code>		
<code> nal_unit_header_svc_extension() /* specified in Annex G */</code>		
<code> nalUnitHeaderBytes += 3</code>		
<code>}</code>		
<code> for(i = nalUnitHeaderBytes; i < NumBytesInNALunit; i++) {</code>		
<code> if(i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {</code>		
<code> rbsp_byte[NumBytesInRBSP++]</code>	All	<code>b(8)</code>
<code> rbsp_byte[NumBytesInRBSP++]</code>	All	<code>b(8)</code>
<code> i += 2</code>		
<code> emulation_prevention_three_byte /* equal to 0x03 */</code>	All	<code>f(8)</code>
<code> } else</code>		
<code> rbsp_byte[NumBytesInRBSP++]</code>	All	<code>b(8)</code>
<code> }</code>		
<code> }</code>		

7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

7.3.2.1 Sequence parameter set RBSP syntax

<code>seq_parameter_set_rbsp() {</code>	C	Descriptor
<code> seq_parameter_set_data()</code>	0	
<code> rbsp_trailing_bits()</code>	0	
<code>}</code>		

7.3.2.1.1 Sequence parameter set data syntax

seq_parameter_set_data() {	C	Descriptor
profile_idc	0	u(8)
constraint_set0_flag	0	u(1)
constraint_set1_flag	0	u(1)
constraint_set2_flag	0	u(1)
constraint_set3_flag	0	u(1)
reserved_zero_4bits /* equal to 0 */	0	u(4)
level_idc	0	u(8)
seq_parameter_set_id	0	ue(v)
if(profile_idc == 100 profile_idc == 110 profile_idc == 122 profile_idc == 244 profile_idc == 44 profile_idc == 83 profile_idc == 86) {		
chroma_format_idc	0	ue(v)
if(chroma_format_idc == 3)		
separate_colour_plane_flag	0	u(1)
bit_depth_luma_minus8	0	ue(v)
bit_depth_chroma_minus8	0	ue(v)
qpprime_y_zero_transform_bypass_flag	0	u(1)
seq_scaling_matrix_present_flag	0	u(1)
if(seq_scaling_matrix_present_flag)		
for(i = 0; i < ((chroma_format_idc != 3) ? 8 : 12); i++) {		
seq_scaling_list_present_flag[i]	0	u(1)
if(seq_scaling_list_present_flag[i])		
if(i < 6)		
scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	0	
else		
scaling_list(ScalingList8x8[i - 6], 64, UseDefaultScalingMatrix8x8Flag[i - 6])	0	
}		
}		
log2_max_frame_num_minus4	0	ue(v)
pic_order_cnt_type	0	ue(v)
if(pic_order_cnt_type == 0)		
log2_max_pic_order_cnt_lsb_minus4	0	ue(v)
else if(pic_order_cnt_type == 1) {		
delta_pic_order_always_zero_flag	0	u(1)
offset_for_non_ref_pic	0	se(v)
offset_for_top_to_bottom_field	0	se(v)
num_ref_frames_in_pic_order_cnt_cycle	0	ue(v)
for(i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++)		
offset_for_ref_frame[i]	0	se(v)
}		
num_ref_frames	0	ue(v)
gaps_in_frame_num_value_allowed_flag	0	u(1)
pic_width_in_mbs_minus1	0	ue(v)
pic_height_in_map_units_minus1	0	ue(v)
frame_mbs_only_flag	0	u(1)

if(!frame_mbs_only_flag)		
mb_adaptive_frame_field_flag	0	u(1)
direct_8x8_inference_flag	0	u(1)
frame_cropping_flag	0	u(1)
if(frame_cropping_flag) {		
frame_crop_left_offset	0	ue(v)
frame_crop_right_offset	0	ue(v)
frame_crop_top_offset	0	ue(v)
frame_crop_bottom_offset	0	ue(v)
}		
vui_parameters_present_flag	0	u(1)
if(vui_parameters_present_flag)		
vui_parameters()	0	
}		

7.3.2.1.1.1 Scaling list syntax

scaling_list(scalingList, sizeofScalingList, useDefaultScalingMatrixFlag) {	C	Descriptor
lastScale = 8		
nextScale = 8		
for(j = 0; j < sizeofScalingList; j++) {		
if(nextScale != 0) {		
delta_scale	0 1	se(v)
nextScale = (lastScale + delta_scale + 256) % 256		
useDefaultScalingMatrixFlag = (j == 0 && nextScale == 0)		
}		
scalingList[j] = (nextScale == 0) ? lastScale : nextScale		
lastScale = scalingList[j]		
}		
}		

7.3.2.1.2 Sequence parameter set extension RBSP syntax

seq_parameter_set_extension_rbsp() {	C	Descriptor
seq_parameter_set_id	10	ue(v)
aux_format_idc	10	ue(v)
if(aux_format_idc != 0) {		
bit_depth_aux_minus8	10	ue(v)
alpha_incr_flag	10	u(1)
alpha_opaque_value	10	u(v)
alpha_transparent_value	10	u(v)
}		
additional_extension_flag	10	u(1)
rbsp_trailing_bits()	10	
}		

7.3.2.2 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	C	Descriptor
pic_parameter_set_id	1	ue(v)
seq_parameter_set_id	1	ue(v)
entropy_coding_mode_flag	1	u(1)
pic_order_present_flag	1	u(1)
num_slice_groups_minus1	1	ue(v)
if(num_slice_groups_minus1 > 0) {		
slice_group_map_type	1	ue(v)
if(slice_group_map_type == 0)		
for(iGroup = 0; iGroup <= num_slice_groups_minus1; iGroup++)		
run_length_minus1[iGroup]	1	ue(v)
else if(slice_group_map_type == 2)		
for(iGroup = 0; iGroup < num_slice_groups_minus1; iGroup++) {		
top_left[iGroup]	1	ue(v)
bottom_right[iGroup]	1	ue(v)
}		
else if(slice_group_map_type == 3		
slice_group_map_type == 4		
slice_group_map_type == 5) {		
slice_group_change_direction_flag	1	u(1)
slice_group_change_rate_minus1	1	ue(v)
} else if(slice_group_map_type == 6) {		
pic_size_in_map_units_minus1	1	ue(v)
for(i = 0; i <= pic_size_in_map_units_minus1; i++)		
slice_group_id[i]	1	u(v)
}		
}		
num_ref_idx_l0_active_minus1	1	ue(v)
num_ref_idx_l1_active_minus1	1	ue(v)
weighted_pred_flag	1	u(1)
weighted_bipred_idc	1	u(2)
pic_init_qp_minus26 /* relative to 26 */	1	se(v)
pic_init_qs_minus26 /* relative to 26 */	1	se(v)
chroma_qp_index_offset	1	se(v)
deblocking_filter_control_present_flag	1	u(1)
constrained_intra_pred_flag	1	u(1)
redundant_pic_cnt_present_flag	1	u(1)
if(more_rbsp_data()) {		
transform_8x8_mode_flag	1	u(1)
pic_scaling_matrix_present_flag	1	u(1)
if(pic_scaling_matrix_present_flag)		
for(i = 0; i < 6 +		
((chroma_format_idc != 3) ? 2 : 6) * transform_8x8_mode_flag;		
i++) {		
pic_scaling_list_present_flag[i]	1	u(1)
if(pic_scaling_list_present_flag[i])		
if(i < 6)		

scaling_list(ScalingList4x4[i], 16, UseDefaultScalingMatrix4x4Flag[i])	1	
else		
scaling_list(ScalingList8x8[i – 6], 64, UseDefaultScalingMatrix8x8Flag[i – 6])	1	
}		
second_chroma_qp_index_offset	1	se(v)
}		
rbp_trailing_bits()	1	
}		

7.3.2.3 Supplemental enhancement information RBSP syntax

sei_rbsp() {	C	Descriptor
do		
sei_message()	5	
while(more_rbsp_data())		
rbp_trailing_bits()	5	
}		

7.3.2.3.1 Supplemental enhancement information message syntax

sei_message() {	C	Descriptor
payloadType = 0		
while(next_bits(8) == 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadType += 255		
}		
last_payload_type_byte	5	u(8)
payloadType += last_payload_type_byte		
payloadSize = 0		
while(next_bits(8) == 0xFF) {		
ff_byte /* equal to 0xFF */	5	f(8)
payloadSize += 255		
}		
last_payload_size_byte	5	u(8)
payloadSize += last_payload_size_byte		
sei_payload(payloadType, payloadSize)	5	
}		

7.3.2.4 Access unit delimiter RBSP syntax

access_unit_delimiter_rbsp() {	C	Descriptor
primary_pic_type	6	u(3)
rbp_trailing_bits()	6	
}		

7.3.2.5 End of sequence RBSP syntax

end_of_seq_rbsp() {	C	Descriptor
}		

7.3.2.6 End of stream RBSP syntax

end_of_stream_rbsp() {	C	Descriptor
}		

7.3.2.7 Filler data RBSP syntax

filler_data_rbsp() {	C	Descriptor
while(next_bits(8) == 0xFF)		
ff_byte /* equal to 0xFF */	9	f(8)
rbsp_trailing_bits()	9	
}		

7.3.2.8 Slice layer without partitioning RBSP syntax

slice_layer_without_partitioning_rbsp() {	C	Descriptor
slice_header()	2	
slice_data() /* all categories of slice_data() syntax */	2 3 4	
rbsp_slice_trailing_bits()	2	
}		

7.3.2.9 Slice data partition RBSP syntax

7.3.2.9.1 Slice data partition A RBSP syntax

slice_data_partition_a_layer_rbsp() {	C	Descriptor
slice_header()	2	
slice_id	All	ue(v)
slice_data() /* only category 2 parts of slice_data() syntax */	2	
rbsp_slice_trailing_bits()	2	
}		

7.3.2.9.2 Slice data partition B RBSP syntax

slice_data_partition_b_layer_rbsp() {	C	Descriptor
slice_id	All	ue(v)
if(separate_colour_plane_flag == 1)		
colour_plane_id	All	u(2)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	All	ue(v)
slice_data() /* only category 3 parts of slice_data() syntax */	3	
rbsp_slice_trailing_bits()	3	
}		

7.3.2.9.3 Slice data partition C RBSP syntax

slice_data_partition_c_layer_rbsp() {	C	Descriptor
slice_id	All	ue(v)
if(separate_colour_plane_flag == 1)		
colour_plane_id	All	u(2)
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	All	ue(v)
slice_data() /* only category 4 parts of slice_data() syntax */	4	
rbsp_slice_trailing_bits()	4	
}		

7.3.2.10 RBSP slice trailing bits syntax

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

7.3.2.11 RBSP trailing bits syntax

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

7.3.3 Slice header syntax

slice_header() {	C	Descriptor
first_mb_in_slice	2	ue(v)
slice_type	2	ue(v)
pic_parameter_set_id	2	ue(v)
if(separate_colour_plane_flag == 1)		
colour_plane_id	2	u(2)
frame_num	2	u(v)
if(!frame_mbs_only_flag) {		
field_pic_flag	2	u(1)
if(field_pic_flag)		
bottom_field_flag	2	u(1)
}		
if(nal_unit_type == 5)		
idr_pic_id	2	ue(v)
if(pic_order_cnt_type == 0) {		
pic_order_cnt_lsb	2	u(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt_bottom	2	se(v)
}		
if(pic_order_cnt_type == 1 && !delta_pic_order_always_zero_flag) {		
delta_pic_order_cnt[0]	2	se(v)
if(pic_order_present_flag && !field_pic_flag)		
delta_pic_order_cnt[1]	2	se(v)
}		
if(redundant_pic_cnt_present_flag)		
redundant_pic_cnt	2	ue(v)
if(slice_type == B)		
direct_spatial_mv_pred_flag	2	u(1)
if(slice_type == P slice_type == SP slice_type == B) {		
num_ref_idx_active_override_flag	2	u(1)
if(num_ref_idx_active_override_flag) {		
num_ref_idx_l0_active_minus1	2	ue(v)
if(slice_type == B)		
num_ref_idx_l1_active_minus1	2	ue(v)
}		
}		
ref_pic_list_reordering()	2	
if((weighted_pred_flag && (slice_type == P slice_type == SP)) (weighted_bipred_idc == 1 && slice_type == B))		
pred_weight_table()	2	
if(nal_ref_idc != 0)		
dec_ref_pic_marking()	2	
if(entropy_coding_mode_flag && slice_type != I && slice_type != SI)		
cabac_init_idc	2	ue(v)
slice_qp_delta	2	se(v)
if(slice_type == SP slice_type == SI) {		

if(slice_type == SP)		
sp_for_switch_flag	2	u(1)
slice_qs_delta	2	se(v)
}		
if(deblocking_filter_control_present_flag) {		
disable_deblocking_filter_idc	2	ue(v)
if(disable_deblocking_filter_idc != 1) {		
slice_alpha_c0_offset_div2	2	se(v)
slice_beta_offset_div2	2	se(v)
}		
}		
if(num_slice_groups_minus1 > 0 && slice_group_map_type >= 3 && slice_group_map_type <= 5)		
slice_group_change_cycle	2	u(v)
}		

7.3.3.1 Reference picture list reordering syntax

ref_pic_list_reordering() {	C	Descriptor
if(slice_type % 5 != 2 && slice_type % 5 != 4) {		
ref_pic_list_reordering_flag_l0	2	u(1)
if(ref_pic_list_reordering_flag_l0)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc == 0 reordering_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc == 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
if(slice_type % 5 == 1) {		
ref_pic_list_reordering_flag_l1	2	u(1)
if(ref_pic_list_reordering_flag_l1)		
do {		
reordering_of_pic_nums_idc	2	ue(v)
if(reordering_of_pic_nums_idc == 0 reordering_of_pic_nums_idc == 1)		
abs_diff_pic_num_minus1	2	ue(v)
else if(reordering_of_pic_nums_idc == 2)		
long_term_pic_num	2	ue(v)
} while(reordering_of_pic_nums_idc != 3)		
}		
}		

7.3.3.2 Prediction weight table syntax

pred_weight_table() {	C	Descriptor
luma_log2_weight_denom	2	ue(v)
if(ChromaArrayType != 0)		
chroma_log2_weight_denom	2	ue(v)
for(i = 0; i <= num_ref_idx_l0_active_minus1; i++) {		
luma_weight_l0_flag	2	u(1)
if(luma_weight_l0_flag) {		
luma_weight_l0[i]	2	se(v)
luma_offset_l0[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l0_flag	2	u(1)
if(chroma_weight_l0_flag)		
for(j = 0; j < 2; j++) {		
chroma_weight_l0[i][j]	2	se(v)
chroma_offset_l0[i][j]	2	se(v)
}		
}		
}		
if(slice_type % 5 == 1)		
for(i = 0; i <= num_ref_idx_l1_active_minus1; i++) {		
luma_weight_l1_flag	2	u(1)
if(luma_weight_l1_flag) {		
luma_weight_l1[i]	2	se(v)
luma_offset_l1[i]	2	se(v)
}		
if(ChromaArrayType != 0) {		
chroma_weight_l1_flag	2	u(1)
if(chroma_weight_l1_flag)		
for(j = 0; j < 2; j++) {		
chroma_weight_l1[i][j]	2	se(v)
chroma_offset_l1[i][j]	2	se(v)
}		
}		
}		
}		

7.3.3.3 Decoded reference picture marking syntax

	C	Descriptor
dec_ref_pic_marking() {		
if(IdrPicFlag) {		
no_output_of_prior_pics_flag	2 5	u(1)
long_term_reference_flag	2 5	u(1)
} else {		
adaptive_ref_pic_marking_mode_flag	2 5	u(1)
if(adaptive_ref_pic_marking_mode_flag)		
do {		
memory_management_control_operation	2 5	ue(v)
if(memory_management_control_operation == 1 memory_management_control_operation == 3)		
difference_of_pic_nums_minus1	2 5	ue(v)
if(memory_management_control_operation == 2)		
long_term_pic_num	2 5	ue(v)
if(memory_management_control_operation == 3 memory_management_control_operation == 6)		
long_term_frame_idx	2 5	ue(v)
if(memory_management_control_operation == 4)		
max_long_term_frame_idx_plus1	2 5	ue(v)
} while(memory_management_control_operation != 0)		
}		
}		

7.3.4 Slice data syntax

slice_data() {	C	Descriptor
if(entropy_coding_mode_flag)		
while(!byte_aligned())		
cabac_alignment_one_bit	2	f(1)
CurrMbAddr = first_mb_in_slice * (1 + MbaffFrameFlag)		
moreDataFlag = 1		
prevMbSkipped = 0		
do {		
if(slice_type != I && slice_type != SI)		
if(!entropy_coding_mode_flag) {		
mb_skip_run	2	ue(v)
prevMbSkipped = (mb_skip_run > 0)		
for(i=0; i<mb_skip_run; i++)		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
moreDataFlag = more_rbsp_data()		
} else {		
mb_skip_flag	2	ae(v)
moreDataFlag = !mb_skip_flag		
}		
if(moreDataFlag) {		
if(MbaffFrameFlag && (CurrMbAddr % 2 == 0 (CurrMbAddr % 2 == 1 && prevMbSkipped)))		
mb_field_decoding_flag	2	u(1) ae(v)
macroblock_layer()	2 3 4	
}		
if(!entropy_coding_mode_flag)		
moreDataFlag = more_rbsp_data()		
else {		
if(slice_type != I && slice_type != SI)		
prevMbSkipped = mb_skip_flag		
if(MbaffFrameFlag && CurrMbAddr % 2 == 0)		
moreDataFlag = 1		
else {		
end_of_slice_flag	2	ae(v)
moreDataFlag = !end_of_slice_flag		
}		
}		
CurrMbAddr = NextMbAddress(CurrMbAddr)		
} while(moreDataFlag)		
}		

7.3.5 Macroblock layer syntax

macroblock_layer() {	C	Descriptor
mb_type	2	ue(v) ae(v)
if(mb_type == I_PCM) {		
while(!byte_aligned())		
pcm_alignment_zero_bit	3	f(1)
for(i = 0; i < 256; i++)		
pcm_sample_luma[i]	3	u(v)
for(i = 0; i < 2 * MbWidthC * MbHeightC; i++)		
pcm_sample_chroma[i]	3	u(v)
} else {		
noSubMbPartSizeLessThan8x8Flag = 1		
if(mb_type != I_NxN && MbPartPredMode(mb_type, 0) != Intra_16x16 && NumMbPart(mb_type) == 4) {		
sub_mb_pred(mb_type)	2	
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8) {		
if(NumSubMbPart(sub_mb_type[mbPartIdx]) > 1)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else if(!direct_8x8_inference_flag)		
noSubMbPartSizeLessThan8x8Flag = 0		
} else {		
if(transform_8x8_mode_flag && mb_type == I_NxN)		
transform_size_8x8_flag	2	u(1) ae(v)
mb_pred(mb_type)	2	
}		
if(MbPartPredMode(mb_type, 0) != Intra_16x16) {		
coded_block_pattern	2	me(v) ae(v)
if(CodedBlockPatternLuma > 0 && transform_8x8_mode_flag && mb_type != I_NxN && noSubMbPartSizeLessThan8x8Flag && (mb_type != B_Direct_16x16 direct_8x8_inference_flag))		
transform_size_8x8_flag	2	u(1) ae(v)
}		
if(CodedBlockPatternLuma > 0 CodedBlockPatternChroma > 0 MbPartPredMode(mb_type, 0) == Intra_16x16) {		
mb_qp_delta	2	se(v) ae(v)
residual(0, 0, 15)	3 4	
}		
}		
}		

7.3.5.1 Macroblock prediction syntax

mb_pred(mb_type) {	C	Descriptor
if(MbPartPredMode(mb_type, 0) == Intra_4x4 MbPartPredMode(mb_type, 0) == Intra_8x8 MbPartPredMode(mb_type, 0) == Intra_16x16) {		
if(MbPartPredMode(mb_type, 0) == Intra_4x4)		
for(luma4x4BlkIdx=0; luma4x4BlkIdx<16; luma4x4BlkIdx++) {		
prev_intra4x4_pred_mode_flag [luma4x4BlkIdx]	2	u(1) ae(v)
if(!prev_intra4x4_pred_mode_flag[luma4x4BlkIdx])		
rem_intra4x4_pred_mode [luma4x4BlkIdx]	2	u(3) ae(v)
}		
if(MbPartPredMode(mb_type, 0) == Intra_8x8)		
for(luma8x8BlkIdx=0; luma8x8BlkIdx<4; luma8x8BlkIdx++) {		
prev_intra8x8_pred_mode_flag [luma8x8BlkIdx]	2	u(1) ae(v)
if(!prev_intra8x8_pred_mode_flag[luma8x8BlkIdx])		
rem_intra8x8_pred_mode [luma8x8BlkIdx]	2	u(3) ae(v)
}		
if(ChromaArrayType == 1 ChromaArrayType == 2)		
intra_chroma_pred_mode	2	ue(v) ae(v)
} else if(MbPartPredMode(mb_type, 0) != Direct) {		
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
ref_idx_l0 [mbPartIdx]	2	te(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 mb_field_decoding_flag) && MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
ref_idx_l1 [mbPartIdx]	2	te(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L1)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0 [mbPartIdx][0][compIdx]	2	se(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < NumMbPart(mb_type); mbPartIdx++)		
if(MbPartPredMode(mb_type, mbPartIdx) != Pred_L0)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1 [mbPartIdx][0][compIdx]	2	se(v) ae(v)
}		
}		

7.3.5.2 Sub-macroblock prediction syntax

sub_mb_pred(mb_type) {	C	Descriptor
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
sub_mb_type [mbPartIdx]	2	ue(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l0_active_minus1 > 0 mb_field_decoding_flag) && mb_type != P_8x8ref0 && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
ref_idx_l0 [mbPartIdx]	2	te(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if((num_ref_idx_l1_active_minus1 > 0 mb_field_decoding_flag) && sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
ref_idx_l1 [mbPartIdx]	2	te(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L1)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l0 [mbPartIdx][subMbPartIdx][compIdx]	2	se(v) ae(v)
for(mbPartIdx = 0; mbPartIdx < 4; mbPartIdx++)		
if(sub_mb_type[mbPartIdx] != B_Direct_8x8 && SubMbPredMode(sub_mb_type[mbPartIdx]) != Pred_L0)		
for(subMbPartIdx = 0; subMbPartIdx < NumSubMbPart(sub_mb_type[mbPartIdx]); subMbPartIdx++)		
for(compIdx = 0; compIdx < 2; compIdx++)		
mvd_l1 [mbPartIdx][subMbPartIdx][compIdx]	2	se(v) ae(v)
}		

7.3.5.3 Residual data syntax

	C	Descriptor
residual(bmFlag, startIdx, endIdx) {		
if(!entropy_coding_mode_flag)		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
residual_luma(i16x16DClevel, i16x16AClevel, level, level8x8,		
bmFlag, startIdx, endIdx)		
Intra16x16DCLevel = i16x16DClevel		
Intra16x16ACLevel = i16x16AClevel		
LumaLevel = level		
LumaLevel8x8 = level8x8		
if(ChromaArrayType == 1 ChromaArrayType == 2) {		
NumC8x8 = 4 / (SubWidthC * SubHeightC)		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
if((CodedBlockPatternChroma & 3) && startIdx == 0)		
/* chroma DC residual present */		
residual_block(ChromaDCLevel[iCbCr], 0, 4 * NumC8x8 - 1,	3 4	
4 * NumC8x8)		
else		
for(i = 0; i < 4 * NumC8x8; i++)		
ChromaDCLevel[iCbCr][i] = 0		
for(iCbCr = 0; iCbCr < 2; iCbCr++)		
for(i8x8 = 0; i8x8 < NumC8x8; i8x8++)		
for(i4x4 = 0; i4x4 < 4; i4x4++)		
if((CodedBlockPatternChroma & 2) && endIdx > 0)		
/* chroma AC residual present */		
residual_block(ChromaACLevel[iCbCr][i8x8*4+i4x4],	3 4	
max(0, startIdx - 1), endIdx - 1, 15)		
else		
for(i = 0; i < 15; i++)		
ChromaACLevel[iCbCr][i8x8*4+i4x4][i] = 0		
} else if(ChromaArrayType == 3) {		
residual_luma(i16x16DClevel, i16x16AClevel, level, level8x8,		
bmFlag, startIdx, endIdx)		
CbIntra16x16DCLevel = i16x16DClevel		
CbIntra16x16ACLevel = i16x16AClevel		
CbLevel = level		
CbLevel8x8 = level8x8		
residual_luma(i16x16DClevel, i16x16AClevel, level, level8x8,		
bmFlag, startIdx, endIdx)		
CrIntra16x16DCLevel = i16x16DClevel		
CrIntra16x16ACLevel = i16x16AClevel		
CrLevel = level		
CrLevel8x8 = level8x8		
}		

7.3.5.3.1 Residual luma syntax

	C	Descriptor
residual_luma(i16x16DClevel, i16x16AClevel, level, level8x8, bmFlag, startIdx, endIdx) {		
if(!entropy_coding_mode_flag)		
residual_block = residual_block_cavlc		
else		
residual_block = residual_block_cabac		
if(!bmFlag && startIdx == 0 && MbPartPredMode(mb_type, 0) == Intra_16x16)		
residual_block(i16x16DClevel, 0, 15, 16)	3	
for(i8x8 = 0; i8x8 < 4; i8x8++)		
if(!transform_size_8x8_flag !entropy_coding_mode_flag)		
for(i4x4 = 0; i4x4 < 4; i4x4++) {		
if(CodedBlockPatternLuma & (1 << i8x8))		
if(!bmFlag && endIdx > 0 && MbPartPredMode(mb_type, 0) == Intra_16x16)		
residual_block(i16x16AClevel[i8x8*4+ i4x4], max(0, startIdx - 1), endIdx - 1, 15)	3	
else		
residual_block(level[i8x8 * 4 + i4x4], startIdx, endIdx, 16)	3 4	
else if(!bmFlag && MbPartPredMode(mb_type, 0) == Intra_16x16)		
for(i = 0; i < 15; i++)		
i16x16AClevel[i8x8 * 4 + i4x4][i] = 0		
else		
for(i = 0; i < 16; i++)		
level[i8x8 * 4 + i4x4][i] = 0		
if(!entropy_coding_mode_flag && transform_size_8x8_flag)		
for(i = 0; i < 16; i++)		
level8x8[i8x8][4 * i + i4x4] = level[i8x8 * 4 + i4x4][i]		
}		
else if(CodedBlockPatternLuma & (1 << i8x8))		
residual_block(level8x8[i8x8], 4 * startIdx, 4 * endIdx + 3, 64)	3 4	
else		
for(i = 0; i < 64; i++)		
level8x8[i8x8][i] = 0		
}		

7.3.5.3.2 Residual block CAVLC syntax

residual_block_cavlc(coeffLevel, startIdx, endIdx, maxNumCoeff) {	C	Descriptor
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
coeff_token	3 4	ce(v)
if(TotalCoeff(coeff_token) > 0) {		
if(TotalCoeff(coeff_token) > 10 && TrailingOnes(coeff_token) < 3)		
suffixLength = 1		
else		
suffixLength = 0		
for(i = 0; i < TotalCoeff(coeff_token); i++)		
if(i < TrailingOnes(coeff_token)) {		
trailing_ones_sign_flag	3 4	u(1)
level[i] = 1 - 2 * trailing_ones_sign_flag		
} else {		
level_prefix	3 4	ce(v)
levelCode = (Min(15, level_prefix) << suffixLength)		
if(suffixLength > 0 level_prefix >= 14) {		
level_suffix	3 4	u(v)
levelCode += level_suffix		
}		
if(level_prefix >= 15 && suffixLength == 0)		
levelCode += 15		
if(level_prefix >= 16)		
levelCode += (1 << (level_prefix - 3)) - 4096		
if(i == TrailingOnes(coeff_token) && TrailingOnes(coeff_token) < 3)		
levelCode += 2		
if(levelCode % 2 == 0)		
level[i] = (levelCode + 2) >> 1		
else		
level[i] = (-levelCode - 1) >> 1		
if(suffixLength == 0)		
suffixLength = 1		
if(Abs(level[i]) > (3 << (suffixLength - 1)) && suffixLength < 6)		
suffixLength++		
}		
if(TotalCoeff(coeff_token) < endIdx - startIdx + 1) {		
total_zeros	3 4	ce(v)
zerosLeft = total_zeros		
} else		
zerosLeft = 0		
for(i = 0; i < TotalCoeff(coeff_token) - 1; i++) {		
if(zerosLeft > 0) {		
run_before	3 4	ce(v)
run[i] = run_before		
} else		
run[i] = 0		

zerosLeft = zerosLeft - run[i]		
}		
run[TotalCoeff(coeff_token) - 1] = zerosLeft		
coeffNum = -1		
for(i = TotalCoeff(coeff_token) - 1; i >= 0; i--) {		
coeffNum += run[i] + 1		
coeffLevel[startIdx + coeffNum] = level[i]		
}		
}		
}		

7.3.5.3.3 Residual block CABAC syntax

residual_block_cabac(coeffLevel, startIdx, endIdx, maxNumCoeff) {	C	Descriptor
if(maxNumCoeff == 64 (ChromaArrayType == 3))		
coded_block_flag	3 4	ae(v)
for(i = 0; i < maxNumCoeff; i++)		
coeffLevel[i] = 0		
if(coded_block_flag) {		
numCoeff = endIdx + 1		
i = startIdx		
do {		
significant_coeff_flag[i]	3 4	ae(v)
if(significant_coeff_flag[i]) {		
last_significant_coeff_flag[i]	3 4	ae(v)
if(last_significant_coeff_flag[i])		
numCoeff = i + 1		
}		
i++		
} while(i < numCoeff - 1)		
coeff_abs_level_minus1[numCoeff - 1]	3 4	ae(v)
coeff_sign_flag[numCoeff - 1]	3 4	ae(v)
coeffLevel[numCoeff - 1] = (coeff_abs_level_minus1[numCoeff - 1] + 1) * (1 - 2 * coeff_sign_flag[numCoeff - 1])		
for(i = numCoeff - 2; i >= 0; i--)		
if(significant_coeff_flag[i]) {		
coeff_abs_level_minus1[i]	3 4	ae(v)
coeff_sign_flag[i]	3 4	ae(v)
coeffLevel[i] = (coeff_abs_level_minus1[i] + 1) * (1 - 2 * coeff_sign_flag[i])		
}		
}		
}		

7.4 Semantics

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in this subclause. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this Recommendation | International Standard.

7.4.1 NAL unit semantics

NOTE 1 – The VCL is specified to efficiently represent the content of the video data. The NAL is specified to format that data and provide header information in a manner appropriate for conveyance on a variety of communication channels or storage media. All data are contained in NAL units, each of which contains an integer number of bytes. A NAL unit specifies a generic format for use in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and byte stream is identical except that each NAL unit can be preceded by a start code prefix and extra padding bytes in the byte stream format.

NumBytesInNALunit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit. Some form of demarcation of NAL unit boundaries is necessary to enable inference of NumBytesInNALunit. One such demarcation method is specified in Annex B for the byte stream format. Other methods of demarcation may be specified outside of this Recommendation | International Standard.

forbidden_zero_bit shall be equal to 0.

nal_ref_idc not equal to 0 specifies that the content of the NAL unit contains a sequence parameter set, a sequence parameter set extension, a subset sequence parameter set, a picture parameter set, a slice of a reference picture, a slice data partition of a reference picture, or a prefix NAL unit preceding a slice of a reference picture.

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, **nal_ref_idc** equal to 0 for a NAL unit containing a slice or slice data partition indicates that the slice or slice data partition is part of a non-reference picture.

nal_ref_idc shall not be equal to 0 for sequence parameter set or sequence parameter set extension or subset sequence parameter set or picture parameter set NAL units. When **nal_ref_idc** is equal to 0 for one NAL unit with **nal_unit_type** in the range of 1 to 4, inclusive, of a particular picture, it shall be equal to 0 for all NAL units with **nal_unit_type** in the range of 1 to 4, inclusive, of the picture.

nal_ref_idc shall not be equal to 0 for NAL units with **nal_unit_type** equal to 5.

nal_ref_idc shall be equal to 0 for all NAL units having **nal_unit_type** equal to 6, 9, 10, 11, or 12.

nal_unit_type specifies the type of RBSP data structure contained in the NAL unit as specified in Table 7-1.

The column marked "C" in Table 7-1 lists the categories of the syntax elements that may be present in the NAL unit. In addition, syntax elements with syntax category "All" may be present, as determined by the syntax and semantics of the RBSP data structure. The presence or absence of any syntax elements of a particular listed category is determined from the syntax and semantics of the associated RBSP data structure. **nal_unit_type** shall not be equal to 3 or 4 unless at least one syntax element is present in the RBSP data structure having a syntax element category value equal to the value of **nal_unit_type** and not categorized as "All".

For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, VCL and non-VCL NAL units are specified in Table 7-1 in the column labelled "Annex A NAL unit type class". For coded video sequences conforming to one or more of the profiles specified in Annex G that are decoded using the decoding process specified in Annex G, VCL and non-VCL NAL units are specified in Table 7-1 in the column labelled "Annex G NAL unit type class". The entry "suffix dependent" for **nal_unit_type** equal to 14, specifies the following.

If the NAL unit directly following in decoding order a NAL unit with **nal_unit_type** equal to 14 is a NAL unit with **nal_unit_type** equal to 1 or 5, the NAL unit with **nal_unit_type** equal to 14 is a VCL NAL unit.

- Otherwise, if the NAL unit directly following in decoding order a NAL unit with **nal_unit_type** equal to 14 is a NAL unit with **nal_unit_type** equal to 12, the NAL unit with **nal_unit_type** equal to 14 is a non-VCL NAL unit.
- Otherwise (the NAL unit directly following in decoding order a NAL unit with **nal_unit_type** equal to 14 is a NAL unit with **nal_unit_type** not equal to 1, 5, or 12), the NAL unit with **nal_unit_type** equal to 14 is a non-VCL NAL unit and decoders shall ignore (remove from the bitstream and discard) the NAL unit with **nal_unit_type** equal to 14 and the NAL unit directly following in decoding order the NAL unit with **nal_unit_type** equal to 14.

Table 7-1 – NAL unit type codes, syntax element categories, and NAL unit type classes

nal_unit_type	Content of NAL unit and RBSP syntax structure	C	Annex A NAL unit type class	Annex G NAL unit type class
0	Unspecified		non-VCL	non-VCL
1	Coded slice of a non-IDR picture slice_layer_without_partitioning_rbsp()	2, 3, 4	VCL	VCL
2	Coded slice data partition A slice_data_partition_a_layer_rbsp()	2	VCL	not applicable
3	Coded slice data partition B slice_data_partition_b_layer_rbsp()	3	VCL	not applicable
4	Coded slice data partition C slice_data_partition_c_layer_rbsp()	4	VCL	not applicable
5	Coded slice of an IDR picture slice_layer_without_partitioning_rbsp()	2, 3	VCL	VCL
6	Supplemental enhancement information (SEI) sei_rbsp()	5	non-VCL	non-VCL
7	Sequence parameter set seq_parameter_set_rbsp()	0	non-VCL	non-VCL
8	Picture parameter set pic_parameter_set_rbsp()	1	non-VCL	non-VCL
9	Access unit delimiter access_unit_delimiter_rbsp()	6	non-VCL	non-VCL
10	End of sequence end_of_seq_rbsp()	7	non-VCL	non-VCL
11	End of stream end_of_stream_rbsp()	8	non-VCL	non-VCL
12	Filler data filler_data_rbsp()	9	non-VCL	non-VCL
13	Sequence parameter set extension seq_parameter_set_extension_rbsp()	10	non-VCL	non-VCL
14	Prefix NAL unit in scalable extension prefix_nal_unit_rbsp() /* specified in Annex G */	2	non-VCL	suffix dependent
15	Subset sequence parameter set subset_seq_parameter_set_rbsp() /* specified in Annex G */	0	non-VCL	non-VCL
16..18	Reserved		non-VCL	non-VCL
19	Coded slice of an auxiliary coded picture without partitioning slice_layer_without_partitioning_rbsp()	2, 3, 4	non-VCL	non-VCL
20	Coded slice in scalable extension slice_layer_in_scalable_extension_rbsp() /* specified in Annex G */	2, 3, 4	non-VCL	VCL
21..23	Reserved		non-VCL	non-VCL
24..31	Unspecified		non-VCL	non-VCL

NAL units having nal_unit_type equal to 13 or 19 may be discarded by decoders without affecting the decoding process for NAL units having nal_unit_type not equal to 13 or 19 and without affecting conformance to this Recommendation | International Standard.

NAL units having `nal_unit_type` equal to 14, 15, or 20 may be discarded by decoders without affecting the decoding process for NAL units having `nal_unit_type` not equal to 14, 15, or 20 and without affecting conformance to profiles specified in Annex A.

NAL units that use `nal_unit_type` equal to 0 or in the range of 24..31, inclusive, shall not affect the decoding process specified in this Recommendation | International Standard.

NOTE 2 – NAL unit types 0 and 24..31 may be used as determined by the application. No decoding process for these values of `nal_unit_type` is specified in this Recommendation | International Standard.

Decoders shall ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of `nal_unit_type`.

NOTE 3 – This requirement allows future definition of compatible extensions to this Recommendation | International Standard.

In the text, coded slice NAL unit collectively refers to a coded slice of a non-IDR picture NAL unit or to a coded slice of an IDR picture NAL unit. The following flag is specified:

$$\text{IdrPicFlag} = (\text{nal_unit_type} == 5 ? 1 : 0) \quad (7-1)$$

When the value of `nal_unit_type` is equal to 5 for a NAL unit containing a slice of a particular picture, the picture shall not contain NAL units with `nal_unit_type` in the range of 1 to 4, inclusive. For coded video sequences conforming to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, such a picture is referred to as an IDR picture.

NOTE 4 – Slice data partitioning cannot be used for IDR pictures.

rbsp_byte[i] is the *i*-th byte of an RBSP. An RBSP is specified as an ordered sequence of bytes as follows.

The RBSP contains an SODB as follows.

- If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.
- Otherwise, the RBSP contains the SODB as follows.
 - 1) The first byte of the RBSP contains the (most significant, left-most) eight bits of the SODB; the next byte of the RBSP shall contain the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.
 - 2) `rbsp_trailing_bits( )` are present after the SODB as follows:
 - i) The first (most significant, left-most) bits of the final RBSP byte contains the remaining bits of the SODB, (if any)
 - ii) The next bit consists of a single `rbsp_stop_one_bit` equal to 1, and
 - iii) When the `rbsp_stop_one_bit` is not the last bit of a byte-aligned byte, one or more `rbsp_alignment_zero_bit` is present to result in byte alignment.
 - 3) One or more `cabac_zero_word` 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the `rbsp_trailing_bits( )` at the end of the RBSP.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "`_rbsp`" suffix. These structures shall be carried within NAL units as the content of the `rbsp_byte[ i ]` data bytes. The association of the RBSP syntax structures to the NAL units shall be as specified in Table 7-1.

NOTE 5 – When the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the `rbsp_stop_one_bit`, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data necessary for the decoding process is contained in the SODB part of the RBSP.

emulation_prevention_three_byte is a byte equal to 0x03. When an `emulation_prevention_three_byte` is present in the NAL unit, it shall be discarded by the decoding process.

The last byte of the NAL unit shall not be equal to 0x00.

Within the NAL unit, the following three-byte sequences shall not occur at any byte-aligned position:

- 0x000000
- 0x000001
- 0x000002

Within the NAL unit, any four-byte sequence that starts with 0x000003 other than the following sequences shall not occur at any byte-aligned position:

- 0x00000300
- 0x00000301

- 0x00000302
- 0x00000303

NOTE 6 – When `nal_unit_type` is equal to 0, particular care must be exercised in the design of encoders to avoid the presence of the above-listed three-byte and four-byte patterns at the beginning of the NAL unit syntax structure, as the syntax element `emulation_prevention_three_byte` cannot be the third byte of a NAL unit.

7.4.1.1 Encapsulation of an SODB within an RBSP (informative)

This subclause does not form an integral part of this Recommendation | International Standard.

The form of encapsulation of an SODB within an RBSP and the use of the `emulation_prevention_three_byte` for encapsulation of an RBSP within a NAL unit is specified for the following purposes:

- to prevent the emulation of start codes within NAL units while allowing any arbitrary SODB to be represented within a NAL unit,
- to enable identification of the end of the SODB within the NAL unit by searching the RBSP for the `rbp_stop_one_bit` starting at the end of the RBSP, and
- to enable a NAL unit to have a size larger than that of the SODB under some circumstances (using one or more `cabac_zero_word`).

The encoder can produce a NAL unit from an RBSP by the following procedure:

The RBSP data is searched for byte-aligned bits of the following binary patterns:

'00000000 00000000 000000xx' (where xx represents any 2 bit pattern: 00, 01, 10, or 11),

and a byte equal to 0x03 is inserted to replace these bit patterns with the patterns.

'00000000 00000000 00000011 000000xx',

and finally, when the last byte of the RBSP data is equal to 0x00 (which can only occur when the RBSP ends in a `cabac_zero_word`), a final byte equal to 0x03 is appended to the end of the data.

The resulting sequence of bytes is then prefixed with the first byte of the NAL unit containing the indication of the type of RBSP data structure it contains. This results in the construction of the entire NAL unit.

This process can allow any SODB to be represented in a NAL unit while ensuring that

- no byte-aligned start code prefix is emulated within the NAL unit, and
- no sequence of 8 zero-valued bits followed by a start code prefix, regardless of byte-alignment, is emulated within the NAL unit.

7.4.1.2 Order of NAL units and association to coded pictures, access units, and video sequences

This subclause specifies constraints on the order of NAL units in the bitstream.

Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in subclauses 7.3, D.1, and E.1 specifies the decoding order of syntax elements. Decoders conforming to this Recommendation | International Standard shall be capable of receiving NAL units and their syntax elements in decoding order.

7.4.1.2.1 Order of sequence and picture parameter set RBSPs and their activation

This subclause specifies the activation process of picture and sequence parameter sets for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

NOTE 1 – The sequence and picture parameter set mechanism decouples the transmission of infrequently changing information from the transmission of coded macroblock data. Sequence and picture parameter sets may, in some applications, be conveyed "out-of-band" using a reliable transport mechanism.

A picture parameter set RBSP includes parameters that can be referred to by the coded slice NAL units or coded slice data partition A NAL units of one or more coded pictures. Each picture parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one picture parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular picture parameter set RBSP results in the deactivation of the previously-active picture parameter set RBSP (if any).

When a picture parameter set RBSP (with a particular value of `pic_parameter_set_id`) is not active and it is referred to by a coded slice NAL unit or coded slice data partition A NAL unit (using that value of `pic_parameter_set_id`), it is activated. This picture parameter set RBSP is called the active picture parameter set RBSP until it is deactivated by the

activation of another picture parameter set RBSP. A picture parameter set RBSP, with that particular value of `pic_parameter_set_id`, shall be available to the decoding process prior to its activation.

Any picture parameter set NAL unit containing the value of `pic_parameter_set_id` for the active picture parameter set RBSP shall have the same content as that of the active picture parameter set RBSP unless it follows the last VCL NAL unit of a coded picture and precedes the first VCL NAL unit of another coded picture.

A sequence parameter set RBSP includes parameters that can be referred to by one or more picture parameter set RBSPs or one or more SEI NAL units containing a buffering period SEI message. Each sequence parameter set RBSP is initially considered not active at the start of the operation of the decoding process. At most one sequence parameter set RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular sequence parameter set RBSP results in the deactivation of the previously-active sequence parameter set RBSP (if any).

When a sequence parameter set RBSP (with a particular value of `seq_parameter_set_id`) is not already active and it is referred to by activation of a picture parameter set RBSP (using that value of `seq_parameter_set_id`) or is referred to by an SEI NAL unit containing a buffering period SEI message (using that value of `seq_parameter_set_id`), it is activated. This sequence parameter set RBSP is called the active sequence parameter set RBSP until it is deactivated by the activation of another sequence parameter set RBSP. A sequence parameter set RBSP, with that particular value of `seq_parameter_set_id`, shall be available to the decoding process prior to its activation. An activated sequence parameter set RBSP shall remain active for the entire coded video sequence.

NOTE 2 – Because an IDR access unit begins a new coded video sequence and an activated sequence parameter set RBSP must remain active for the entire coded video sequence, a sequence parameter set RBSP can only be activated by a buffering period SEI message when the buffering period SEI message is part of an IDR access unit.

Any sequence parameter set NAL unit containing the value of `seq_parameter_set_id` for the active sequence parameter set RBSP shall have the same content as that of the active sequence parameter set RBSP unless it follows the last access unit of a coded video sequence and precedes the first VCL NAL unit and the first SEI NAL unit containing a buffering period SEI message (when present) of another coded video sequence.

NOTE 3 – If picture parameter set RBSP or sequence parameter set RBSP are conveyed within the bitstream, these constraints impose an order constraint on the NAL units that contain the picture parameter set RBSP or sequence parameter set RBSP, respectively. Otherwise (picture parameter set RBSP or sequence parameter set RBSP are conveyed by other means not specified in this Recommendation | International Standard), they must be available to the decoding process in a timely fashion such that these constraints are obeyed.

When present, a sequence parameter set extension RBSP includes parameters having a similar function to those of a sequence parameter set RBSP. For purposes of establishing constraints on the syntax elements of the sequence parameter set extension RBSP and for purposes of determining activation of a sequence parameter set extension RBSP, the sequence parameter set extension RBSP shall be considered part of the preceding sequence parameter set RBSP with the same value of `seq_parameter_set_id`. When a sequence parameter set RBSP is present that is not followed by a sequence parameter set extension RBSP with the same value of `seq_parameter_set_id` prior to the activation of the sequence parameter set RBSP, the sequence parameter set extension RBSP and its syntax elements shall be considered not present for the active sequence parameter set RBSP.

All constraints that are expressed on the relationship between the values of the syntax elements (and the values of variables derived from those syntax elements) in sequence parameter sets and picture parameter sets and other syntax elements are expressions of constraints that apply only to the active sequence parameter set and the active picture parameter set. If any sequence parameter set RBSP is present that is not activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream. If any picture parameter set RBSP is present that is not ever activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it were activated by reference in an otherwise-conforming bitstream.

During operation of the decoding process (see clause 8), the values of parameters of the active picture parameter set and the active sequence parameter set shall be considered in effect. For interpretation of SEI messages, the values of the parameters of the picture parameter set and sequence parameter set that are active for the operation of the decoding process for the VCL NAL units of the primary coded picture in the same access unit shall be considered in effect unless otherwise specified in the SEI message semantics.

7.4.1.2.2 Order of access units and association to coded video sequences

A bitstream conforming to this Recommendation | International Standard consists of one or more coded video sequences.

A coded video sequence consists of one or more access units. For coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9, the order of NAL units and coded pictures and their association to access units is described in subclause 7.4.1.2.3.

The first access unit of each coded video sequence is an IDR access unit. All subsequent access units in the coded video sequence are non-IDR access units.

The values of picture order count for the coded pictures in consecutive access units in decoding order containing non-reference pictures shall be non-decreasing.

When present, an access unit following an access unit that contains an end of sequence NAL unit shall be an IDR access unit.

When an SEI NAL unit contains data that pertain to more than one access unit (for example, when the SEI NAL unit has a coded video sequence as its scope), it shall be contained in the first access unit to which it applies.

When an end of stream NAL unit is present in an access unit, this access unit shall be the last access unit in the bitstream and the end of stream NAL unit shall be the last NAL unit in that access unit.

7.4.1.2.3 Order of NAL units and coded pictures and association to access units

This subclause specifies the order of NAL units and coded pictures and association to access unit for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

An access unit consists of one primary coded picture, zero or more corresponding redundant coded pictures, and zero or more non-VCL NAL units. The association of VCL NAL units to primary or redundant coded pictures is described in subclause 7.4.1.2.5.

The first access unit in the bitstream starts with the first NAL unit of the bitstream.

The first of any of the following NAL units after the last VCL NAL unit of a primary coded picture specifies the start of a new access unit.

- access unit delimiter NAL unit (when present)
- sequence parameter set NAL unit (when present)
- picture parameter set NAL unit (when present)
- SEI NAL unit (when present)
- NAL units with nal_unit_type in the range of 14 to 18, inclusive
- first VCL NAL unit of a primary coded picture (always present)

The constraints for the detection of the first VCL NAL unit of a primary coded picture are specified in subclause 7.4.1.2.4.

The following constraints shall be obeyed by the order of the coded pictures and non-VCL NAL units within an access unit.

- When an access unit delimiter NAL unit is present, it shall be the first NAL unit. There shall be at most one access unit delimiter NAL unit in any access unit.
- When any SEI NAL units are present, they shall precede the primary coded picture.
- When an SEI NAL unit containing a buffering period SEI message is present, the buffering period SEI message shall be the first SEI message payload of the first SEI NAL unit in the access unit.
- The primary coded picture shall precede the corresponding redundant coded pictures.

When redundant coded pictures are present, they shall be ordered in ascending order of the value of redundant_pic_cnt.

- When a sequence parameter set extension NAL unit is present, it shall be the next NAL unit after a sequence parameter set NAL unit having the same value of seq_parameter_set_id as in the sequence parameter set extension NAL unit.
- When one or more coded slice of an auxiliary coded picture without partitioning NAL units is present, they shall follow the primary coded picture and all redundant coded pictures (if any).
- When an end of sequence NAL unit is present, it shall follow the primary coded picture and all redundant coded pictures (if any) and all coded slice of an auxiliary coded picture without partitioning NAL units (if any).
- When an end of stream NAL unit is present, it shall be the last NAL unit.

- NAL units having `nal_unit_type` equal to 0, 12, or in the range of 20 to 31, inclusive, shall not precede the first VCL NAL unit of the primary coded picture.

NOTE 1 – Sequence parameter set NAL units or picture parameter set NAL units may be present in an access unit, but cannot follow the last VCL NAL unit of the primary coded picture within the access unit, as this condition would specify the start of a new access unit.

NOTE 2 – When a NAL unit having `nal_unit_type` equal to 7 or 8 is present in an access unit, it may or may not be referred to in the coded pictures of the access unit in which it is present, and may be referred to in coded pictures of subsequent access units.

The structure of access units not containing any NAL units with `nal_unit_type` equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive, is shown in Figure 7-1.

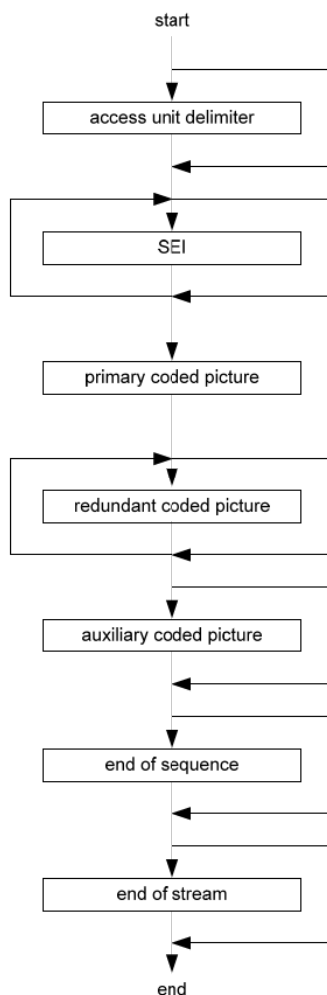


Figure 7-1 – Structure of an access unit not containing any NAL units with `nal_unit_type` equal to 0, 7, 8, or in the range of 12 to 18, inclusive, or in the range of 20 to 31, inclusive

7.4.1.2.4 Detection of the first VCL NAL unit of a primary coded picture

This subclause specifies constraints on VCL NAL unit syntax that are sufficient to enable the detection of the first VCL NAL unit of each primary coded picture for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

Any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the current access unit shall be different from any coded slice NAL unit or coded slice data partition A NAL unit of the primary coded picture of the previous access unit in one or more of the following ways.

- `frame_num` differs in value. The value of `frame_num` used to test this condition is the value of `frame_num` that appears in the syntax of the slice header, regardless of whether that value is inferred to have been equal to 0 for subsequent use in the decoding process due to the presence of `memory_management_control_operation` equal to 5.
NOTE 1 – A consequence of the above statement is that a primary coded picture having `frame_num` equal to 1 cannot contain a `memory_management_control_operation` equal to 5 unless some other condition listed below is fulfilled for the next primary coded picture that follows after it (if any).
- `pic_parameter_set_id` differs in value.
- `field_pic_flag` differs in value.
- `bottom_field_flag` is present in both and differs in value.
- `nal_ref_idc` differs in value with one of the `nal_ref_idc` values being equal to 0.
- `pic_order_cnt_type` is equal to 0 for both and either `pic_order_cnt_lsb` differs in value, or `delta_pic_order_cnt_bottom` differs in value.
- `pic_order_cnt_type` is equal to 1 for both and either `delta_pic_order_cnt[0]` differs in value, or `delta_pic_order_cnt[1]` differs in value.
- `IdrPicFlag` differs in value.
- `IdrPicFlag` is equal to 1 for both and `idr_pic_id` differs in value.

NOTE 2 – Some of the VCL NAL units in redundant coded pictures or some non-VCL NAL units (e.g., an access unit delimiter NAL unit) may also be used for the detection of the boundary between access units, and may therefore aid in the detection of the start of a new primary coded picture.

7.4.1.2.5 Order of VCL NAL units and association to coded pictures

This subclause specifies the order of VCL NAL units and association to coded pictures for coded video sequences that conform to one or more of the profiles specified in Annex A that are decoded using the decoding process specified in clauses 2-9.

Each VCL NAL unit is part of a coded picture.

The order of the VCL NAL units within a coded IDR picture is constrained as follows.

- If arbitrary slice order is allowed as specified in Annex A, coded slice of an IDR picture NAL units may have any order relative to each other.
- Otherwise (arbitrary slice order is not allowed), the following applies.
 - If `separate_colour_plane_flag` is equal to 0, coded slice of an IDR picture NAL units of a slice group shall not be interleaved with coded slice of an IDR picture NAL units of another slice group and the order of coded slice of an IDR picture NAL units within a slice group shall be in the order of increasing macroblock address for the first macroblock of each coded slice of an IDR picture NAL unit of the particular slice group.
 - Otherwise (`separate_colour_plane_flag` is equal to 1), coded slice of an IDR picture NAL units of a slice group for a particular value of `colour_plane_id` shall not be interleaved with coded slice of an IDR picture NAL units of another slice group with the same value of `colour_plane_id` and the order of coded slices of IDR picture NAL units within a slice group for a particular value of `colour_plane_id` shall be in the order of increasing macroblock address for the first macroblock of each coded slice of an IDR picture NAL unit of the particular slice group having the particular value of `colour_plane_id`.

NOTE 1 – When `separate_colour_plane_flag` is equal to 1, the relative ordering of coded slices having different values of `colour_plane_id` is not constrained.

The order of the VCL NAL units within a coded non-IDR picture is constrained as follows.

- If arbitrary slice order is allowed as specified in Annex A, coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units may have any order relative to each other. A coded slice data partition A NAL unit with a particular value of `slice_id` shall precede any present coded slice data partition B NAL unit with the same value of `slice_id`. A coded slice data partition A NAL unit with a particular value of `slice_id` shall precede any present coded slice data partition C NAL unit with the same value of `slice_id`. When a coded slice data partition B NAL unit with a particular value of `slice_id` is present, it shall precede any present coded slice data partition C NAL unit with the same value of `slice_id`.
- Otherwise (arbitrary slice order is not allowed), the following applies.
 - If `separate_colour_plane_flag` is equal to 0, coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of a slice group shall not be interleaved with coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of another slice group and the order of coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units within a slice group shall be in the order of increasing macroblock address for the first macroblock of each coded slice of a non-IDR picture NAL unit or coded slice data partition A NAL unit of the particular slice group. A coded slice data partition A NAL unit with a particular value of `slice_id` shall immediately precede any present coded slice data partition B NAL unit with the same value of `slice_id`. A coded slice data partition A NAL unit with a particular value of `slice_id` shall immediately precede any present coded slice data partition C NAL unit with the same value of `slice_id`, when a coded slice data partition B NAL unit with the same value of `slice_id` is not present. When a coded slice data partition B NAL unit with a particular value of `slice_id` is present, it shall immediately precede any present coded slice data partition C NAL unit with the same value of `slice_id`.
 - Otherwise (`separate_colour_plane_flag` is equal to 1), coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of a slice group for a particular value of `colour_plane_id` shall not be interleaved with coded slice of a non-IDR picture NAL units or coded slice data partition NAL units of another slice group with the same value of `colour_plane_id` and the order of coded slice of a non-IDR picture NAL units or coded slice data partition A NAL units within a slice group for particular value of `colour_plane_id` shall be in the order of increasing macroblock address for the first macroblock of each coded slice of a non-IDR picture NAL unit or coded slice data partition A NAL unit of the particular slice group having the particular value of `colour_plane_id`. A coded slice data partition A NAL unit associated with a particular value of `slice_id` and `colour_plane_id` shall immediately precede any present coded slice data partition B NAL unit with the same value of `slice_id` and `colour_plane_id`. A coded slice data partition A NAL unit associated with a particular value of `slice_id` and `colour_plane_id` shall immediately precede any present coded slice data partition C NAL unit with the same value of `slice_id` and `colour_plane_id`, when a coded slice data partition B NAL unit with the same value of `slice_id` and `colour_plane_id` is not present. When a coded slice data partition B NAL unit with a particular value of `slice_id` and `colour_plane_id` is present, it shall immediately precede any present coded slice data partition C NAL unit with the same value of `slice_id` and `colour_plane_id`.

NOTE 2 – When `separate_colour_plane_flag` is equal to 1, the relative ordering of coded slices having different values of `colour_plane_id` is not constrained.

NAL units having `nal_unit_type` equal to 12 may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having `nal_unit_type` equal to 0 or in the range of 24 to 31, inclusive, which are unspecified, may be present in the access unit but shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

NAL units having `nal_unit_type` in the range of 20 to 23, inclusive, shall not precede the first VCL NAL unit of the primary coded picture within the access unit.

7.4.2 Raw byte sequence payloads and RBSP trailing bits semantics

7.4.2.1 Sequence parameter set RBSP semantics

7.4.2.1.1 Sequence parameter set data semantics

`profile_idc` and `level_idc` indicate the profile and level to which the coded video sequence conforms.

`constraint_set0_flag` equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.1. `constraint_set0_flag` equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.1.

`constraint_set1_flag` equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.2. `constraint_set1_flag` equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.2.

constraint_set2_flag equal to 1 indicates that the coded video sequence obeys all constraints specified in subclause A.2.3. **constraint_set2_flag** equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in subclause A.2.3.

NOTE 1 – When one or more than one of **constraint_set0_flag**, **constraint_set1_flag**, or **constraint_set2_flag** are equal to 1, the coded video sequence must obey the constraints of all of the indicated subclauses of subclause A.2. When **profile_idc** is equal to 44, 100, 110, 122, or 244, the values of **constraint_set0_flag**, **constraint_set1_flag**, and **constraint_set2_flag** must all be equal to 0.

constraint_set3_flag indicates the following.

- If **profile_idc** is equal to 66, 77, or 88 and **level_idc** is equal to 11, **constraint_set3_flag** equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for level 1b and **constraint_set3_flag** equal to 0 indicates that the coded video sequence may or may not obey all constraints specified in Annex A for level 1b.
- Otherwise, if **profile_idc** is equal to 100 or 110, **constraint_set3_flag** equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for the High 10 Intra profile, and **constraint_set3_flag** equal to 0 indicates that the coded video sequence may or may not obey these corresponding constraints.
- Otherwise, if **profile_idc** is equal to 122, **constraint_set3_flag** equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for the High 4:2:2 Intra profile, and **constraint_set3_flag** equal to 0 indicates that the coded video sequence may or may not obey these corresponding constraints.
- Otherwise, if **profile_idc** is equal to 44, **constraint_set3_flag** shall be equal to 1. When **profile_idc** is equal to 44, the value of 0 for **constraint_set3_flag** is forbidden.
- Otherwise, if **profile_idc** is equal to 244, **constraint_set3_flag** equal to 1 indicates that the coded video sequence obeys all constraints specified in Annex A for the High 4:4:4 Intra profile, and **constraint_set3_flag** equal to 0 indicates that the coded video sequence may or may not obey these corresponding constraints.
- Otherwise (**profile_idc** is equal to 66, 77, or 88 and **level_idc** is not equal to 11), the value of 1 for **constraint_set3_flag** is reserved for future use by ITU-T | ISO/IEC. **constraint_set3_flag** shall be equal to 0 for coded video sequences with **profile_idc** equal to 66, 77, or 88 and **level_idc** not equal to 11 in bitstreams conforming to this Recommendation | International Standard. Decoders conforming to this Recommendation | International Standard shall ignore the value of **constraint_set3_flag** when **profile_idc** is equal to 66, 77, or 88 and **level_idc** is not equal to 11.

reserved_zero_4bits shall be equal to 0. Other values of **reserved_zero_4bits** may be specified in the future by ITU-T | ISO/IEC. Decoders shall ignore the value of **reserved_zero_4bits**.

seq_parameter_set_id identifies the sequence parameter set that is referred to by the picture parameter set. The value of **seq_parameter_set_id** shall be in the range of 0 to 31, inclusive.

NOTE 2 – When feasible, encoders should use distinct values of **seq_parameter_set_id** when the values of other sequence parameter set syntax elements differ rather than changing the values of the syntax elements associated with a specific value of **seq_parameter_set_id**.

chroma_format_idc specifies the chroma sampling relative to the luma sampling as specified in subclause 6.2. The value of **chroma_format_idc** shall be in the range of 0 to 3, inclusive. When **chroma_format_idc** is not present, it shall be inferred to be equal to 1 (4:2:0 chroma format).

separate_colour_plane_flag equal to 1 specifies that the three colour components of the 4:4:4 chroma format are coded separately. **separate_colour_plane_flag** equal to 0 specifies that the colour components are not coded separately. When **separate_colour_plane_flag** is not present, it shall be inferred to be equal to 0. When **separate_colour_plane_flag** is equal to 1, the primary coded picture consists of three separate components, each of which consists of coded samples of one colour plane (Y, Cb or Cr) that each use the monochrome coding syntax. In this case, each colour plane is associated with a specific **colour_plane_id** value.

NOTE 3 – There is no dependency in decoding processes between the colour planes having different **colour_plane_id** values. For example, the decoding process of a monochrome picture with one value of **colour_plane_id** does not use any data from monochrome pictures having different values of **colour_plane_id** for inter prediction.

Depending on the value of **separate_colour_plane_flag**, the value of the variable **ChromaArrayType** is assigned as follows.

- If **separate_colour_plane_flag** is equal to 0, **ChromaArrayType** is set equal to **chroma_format_idc**.
- Otherwise (**separate_colour_plane_flag** is equal to 1), **ChromaArrayType** is set equal to 0.

bit_depth_luma_minus8 specifies the bit depth of the samples of the luma array and the value of the luma quantisation parameter range offset **QpBdOffset_Y**, as specified by

$$\text{BitDepth}_Y = 8 + \text{bit_depth_luma_minus8} \quad (7-2)$$

$$QpBdOffset_Y = 6 * \text{bit_depth_luma_minus8} \quad (7-3)$$

When `bit_depth_luma_minus8` is not present, it shall be inferred to be equal to 0. `bit_depth_luma_minus8` shall be in the range of 0 to 6, inclusive.

`bit_depth_chroma_minus8` specifies the bit depth of the samples of the chroma arrays and the value of the chroma quantisation parameter range offset `QpBdOffset_C`, as specified by

$$\text{BitDepth}_C = 8 + \text{bit_depth_chroma_minus8} \quad (7-4)$$

$$QpBdOffset_C = 6 * (\text{bit_depth_chroma_minus8} + \text{residual_colour_transform_flag}) \quad (7-5)$$

When `bit_depth_chroma_minus8` is not present, it shall be inferred to be equal to 0. `bit_depth_chroma_minus8` shall be in the range of 0 to 6, inclusive.

NOTE 4 – The value of `bit_depth_chroma_minus8` is not used in the decoding process when `ChromaArrayType` is equal to 0. In particular, when `separate_colour_plane_flag` is equal to 1, each colour plane is decoded as a distinct monochrome picture using the luma component decoding process (except for the selection of scaling matrices) and the luma bit depth is used for all three colour components.

The variable `RawMbBits` is derived as

$$\text{RawMbBits} = 256 * \text{BitDepth}_Y + 2 * \text{MbWidth}_C * \text{MbHeight}_C * \text{BitDepth}_C \quad (7-6)$$

`qpprime_y_zero_transform_bypass_flag` equal to 1 specifies that, when `QP'_Y` is equal to 0, a transform bypass operation for the transform coefficient decoding process and picture construction process prior to deblocking filter process as specified in subclause 8.5 shall be applied. `qpprime_y_zero_transform_bypass_flag` equal to 0 specifies that the transform coefficient decoding process and picture construction process prior to deblocking filter process shall not use the transform bypass operation. When `qpprime_y_zero_transform_bypass_flag` is not present, it shall be inferred to be equal to 0.

`seq_scaling_matrix_present_flag` equal to 1 specifies that the flags `seq_scaling_list_present_flag[i]` for $i = 0..7$ or $i = 0..11$ are present. `seq_scaling_matrix_present_flag` equal to 0 specifies that these flags are not present and the sequence-level scaling list specified by `Flat_4x4_16` shall be inferred for $i = 0..5$ and the sequence-level scaling list specified by `Flat_8x8_16` shall be inferred for $i = 6..11$. When `seq_scaling_matrix_present_flag` is not present, it shall be inferred to be equal to 0.

The scaling lists `Flat_4x4_16` and `Flat_8x8_16` are specified as follows:

$$\text{Flat_4x4_16}[i] = 16, \quad \text{with } i = 0..15, \quad (7-7)$$

$$\text{Flat_8x8_16}[i] = 16, \quad \text{with } i = 0..63. \quad (7-8)$$

`seq_scaling_list_present_flag[i]` equal to 1 specifies that the syntax structure for scaling list i is present in the sequence parameter set. `seq_scaling_list_present_flag[i]` equal to 0 specifies that the syntax structure for scaling list i is not present in the sequence parameter set and the scaling list fall-back rule set A specified in Table 7-2 shall be used to infer the sequence-level scaling list for index i .

Table 7-2 – Assignment of mnemonic names to scaling list indices and specification of fall-back rule

Value of scaling list index	Mnemonic name	Block size	MB prediction type	Component	Scaling list fall-back rule set A	Scaling list fall-back rule set B	Default scaling list
0	Sl_4x4_Intra_Y	4x4	Intra	Y	default scaling list	sequence-level scaling list	Default_4x4_Intra
1	Sl_4x4_Intra_Cb	4x4	Intra	Cb	scaling list for i = 0	scaling list for i = 0	Default_4x4_Intra
2	Sl_4x4_Intra_Cr	4x4	Intra	Cr	scaling list for i = 1	scaling list for i = 1	Default_4x4_Intra
3	Sl_4x4_Inter_Y	4x4	Inter	Y	default scaling list	sequence-level scaling list	Default_4x4_Inter
4	Sl_4x4_Inter_Cb	4x4	Inter	Cb	scaling list for i = 3	scaling list for i = 3	Default_4x4_Inter
5	Sl_4x4_Inter_Cr	4x4	Inter	Cr	scaling list for i = 4	scaling list for i = 4	Default_4x4_Inter
6	Sl_8x8_Intra_Y	8x8	Intra	Y	default scaling list	sequence-level scaling list	Default_8x8_Intra
7	Sl_8x8_Inter_Y	8x8	Inter	Y	default scaling list	sequence-level scaling list	Default_8x8_Inter
8	Sl_8x8_Intra_Cb	8x8	Intra	Cb	scaling list for i = 6	scaling list for i = 6	Default_8x8_Intra
9	Sl_8x8_Inter_Cb	8x8	Inter	Cb	scaling list for i = 7	scaling list for i = 7	Default_8x8_Inter
10	Sl_8x8_Intra_Cr	8x8	Intra	Cr	scaling list for i = 8	scaling list for i = 8	Default_8x8_Intra
11	Sl_8x8_Inter_Cr	8x8	Inter	Cr	scaling list for i = 9	scaling list for i = 9	Default_8x8_Inter

Table 7-3 specifies the default scaling lists Default_4x4_Intra and Default_4x4_Inter. Table 7-4 specifies the default scaling lists Default_8x8_Intra and Default_8x8_Inter.

Table 7-3 – Specification of default scaling lists Default_4x4_Intra and Default_4x4_Inter

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Default_4x4_Intra[idx]	6	13	13	20	20	20	28	28	28	28	32	32	32	37	37	42
Default_4x4_Inter[idx]	10	14	14	20	20	20	24	24	24	24	27	27	27	30	30	34

Table 7-4 – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Default_8x8_Intra[idx]	6	10	10	13	11	13	16	16	16	16	18	18	18	18	18	23
Default_8x8_Inter[idx]	9	13	13	15	13	15	17	17	17	17	19	19	19	19	19	21

Table 7-4 (continued) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Default_8x8_Intra[idx]	23	23	23	23	23	25	25	25	25	25	25	25	27	27	27	27
Default_8x8_Inter[idx]	21	21	21	21	21	22	22	22	22	22	22	22	24	24	24	24

Table 7-4 (continued) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Default_8x8_Intra[idx]	27	27	27	27	29	29	29	29	29	29	29	31	31	31	31	31
Default_8x8_Inter[idx]	24	24	24	24	25	25	25	25	25	25	25	27	27	27	27	27

Table 7-4 (concluded) – Specification of default scaling lists Default_8x8_Intra and Default_8x8_Inter

idx	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Default_8x8_Intra[idx]	31	33	33	33	33	33	36	36	36	36	38	38	38	40	40	42
Default_8x8_Inter[idx]	27	28	28	28	28	28	30	30	30	30	32	32	32	33	33	35

log2_max_frame_num_minus4 specifies the value of the variable MaxFrameNum that is used in frame_num related derivations as follows:

$$\text{MaxFrameNum} = 2^{(\text{log2_max_frame_num_minus4} + 4)} \quad (7-9)$$

The value of log2_max_frame_num_minus4 shall be in the range of 0 to 12, inclusive.

pic_order_cnt_type specifies the method to decode picture order count (as specified in subclause 8.2.1). The value of pic_order_cnt_type shall be in the range of 0 to 2, inclusive.

pic_order_cnt_type shall not be equal to 2 in a coded video sequence that contains any of the following:

- an access unit containing a non-reference frame followed immediately by an access unit containing a non-reference picture;
- two access units each containing a field with the two fields together forming a complementary non-reference field pair followed immediately by an access unit containing a non-reference picture;
- an access unit containing a non-reference field followed immediately by an access unit containing another non-reference picture that does not form a complementary non-reference field pair with the first of the two access units.

log2_max_pic_order_cnt_lsb_minus4 specifies the value of the variable MaxPicOrderCntLsb that is used in the decoding process for picture order count as specified in subclause 8.2.1 as follows:

$$\text{MaxPicOrderCntLsb} = 2^{(\text{log2_max_pic_order_cnt_lsb_minus4} + 4)} \quad (7-10)$$

The value of log2_max_pic_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

delta_pic_order_always_zero_flag equal to 1 specifies that delta_pic_order_cnt[0] and delta_pic_order_cnt[1] are not present in the slice headers of the sequence and shall be inferred to be equal to 0. delta_pic_order_always_zero_flag

equal to 0 specifies that `delta_pic_order_cnt[0]` is present in the slice headers of the sequence and `delta_pic_order_cnt[1]` may be present in the slice headers of the sequence.

offset_for_non_ref_pic is used to calculate the picture order count of a non-reference picture as specified in 8.2.1. The value of `offset_for_non_ref_pic` shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

offset_for_top_to_bottom_field is used to calculate the picture order count of a bottom field as specified in subclause 8.2.1. The value of `offset_for_top_to_bottom_field` shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

num_ref_frames_in_pic_order_cnt_cycle is used in the decoding process for picture order count as specified in subclause 8.2.1. The value of `num_ref_frames_in_pic_order_cnt_cycle` shall be in the range of 0 to 255, inclusive.

offset_for_ref_frame[i] is an element of a list of `num_ref_frames_in_pic_order_cnt_cycle` values used in the decoding process for picture order count as specified in subclause 8.2.1. The value of `offset_for_ref_frame[i]` shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

num_ref_frames specifies the maximum number of short-term and long-term reference frames, complementary reference field pairs, and non-paired reference fields that may be used by the decoding process for inter prediction of any picture in the sequence. `num_ref_frames` also determines the size of the sliding window operation as specified in subclause 8.2.5.3. The value of `num_ref_frames` shall be in the range of 0 to `MaxDpbSize` (as specified in subclause A.3.1 or A.3.2), inclusive.

gaps_in_frame_num_value_allowed_flag specifies the allowed values of `frame_num` as specified in subclause 7.4.3 and the decoding process in case of an inferred gap between values of `frame_num` as specified in subclause 8.2.5.2.

pic_width_in_mbs_minus1 plus 1 specifies the width of each decoded picture in units of macroblocks.

The variable for the picture width in units of macroblocks is derived as follows:

$$\text{PicWidthInMbs} = \text{pic_width_in_mbs_minus1} + 1 \quad (7-11)$$

The variable for picture width for the luma component is derived as follows:

$$\text{PicWidthInSamples}_L = \text{PicWidthInMbs} * 16 \quad (7-12)$$

The variable for picture width for the chroma components is derived as follows:

$$\text{PicWidthInSamples}_C = \text{PicWidthInMbs} * \text{MbWidthC} \quad (7-13)$$

pic_height_in_map_units_minus1 plus 1 specifies the height in slice group map units of a decoded frame or field.

The variables `PicHeightInMapUnits` and `PicSizeInMapUnits` are derived as follows:

$$\text{PicHeightInMapUnits} = \text{pic_height_in_map_units_minus1} + 1 \quad (7-14)$$

$$\text{PicSizeInMapUnits} = \text{PicWidthInMbs} * \text{PicHeightInMapUnits} \quad (7-15)$$

frame_mbs_only_flag equal to 0 specifies that coded pictures of the coded video sequence may either be coded fields or coded frames. `frame_mbs_only_flag` equal to 1 specifies that every coded picture of the coded video sequence is a coded frame containing only frame macroblocks.

The allowed range of values for `pic_width_in_mbs_minus1`, `pic_height_in_map_units_minus1`, and `frame_mbs_only_flag` is specified by constraints in Annex A.

Depending on `frame_mbs_only_flag`, semantics are assigned to `pic_height_in_map_units_minus1` as follows:

- If `frame_mbs_only_flag` is equal to 0, `pic_height_in_map_units_minus1` plus 1 is the height of a field in units of macroblocks.
- Otherwise (`frame_mbs_only_flag` is equal to 1), `pic_height_in_map_units_minus1` plus 1 is the height of a frame in units of macroblocks.

The variable `FrameHeightInMbs` is derived as follows:

$$\text{FrameHeightInMbs} = (2 - \text{frame_mbs_only_flag}) * \text{PicHeightInMapUnits} \quad (7-16)$$

mb_adaptive_frame_field_flag equal to 0 specifies no switching between frame and field macroblocks within a picture. **mb_adaptive_frame_field_flag** equal to 1 specifies the possible use of switching between frame and field macroblocks within frames. When **mb_adaptive_frame_field_flag** is not present, it shall be inferred to be equal to 0.

direct_8x8_inference_flag specifies the method used in the derivation process for luma motion vectors for B_Skip, B_Direct_16x16 and B_Direct_8x8 as specified in subclause 8.4.1.2. When **frame_mbs_only_flag** is equal to 0, **direct_8x8_inference_flag** shall be equal to 1.

frame_cropping_flag equal to 1 specifies that the frame cropping offset parameters follow next in the sequence parameter set. **frame_cropping_flag** equal to 0 specifies that the frame cropping offset parameters are not present.

frame_crop_left_offset, **frame_crop_right_offset**, **frame_crop_top_offset**, **frame_crop_bottom_offset** specify the samples of the pictures in the coded video sequence that are output from the decoding process, in terms of a rectangular region specified in frame coordinates for output.

The variables CropUnitX and CropUnitY are derived as follows:

— If ChromaArrayType is equal to 0, CropUnitX and CropUnitY are derived as

$$\text{CropUnitX} = 1 \quad (7-17)$$

$$\text{CropUnitY} = 2 - \text{frame_mbs_only_flag} \quad (7-18)$$

— Otherwise (ChromaArrayType is equal to 1, 2, or 3), CropUnitX and CropUnitY are derived as

$$\text{CropUnitX} = \text{SubWidthC} \quad (7-19)$$

$$\text{CropUnitY} = \text{SubHeightC} * (2 - \text{frame_mbs_only_flag}) \quad (7-20)$$

The frame cropping rectangle contains luma samples with horizontal frame coordinates from $\text{CropUnitX} * \text{frame_crop_left_offset}$ to $\text{PicWidthInSamples}_L - (\text{CropUnitX} * \text{frame_crop_right_offset} + 1)$ and vertical frame coordinates from $\text{CropUnitY} * \text{frame_crop_top_offset}$ to $(16 * \text{FrameHeightInMbs}) - (\text{CropUnitY} * \text{frame_crop_bottom_offset} + 1)$, inclusive. The value of **frame_crop_left_offset** shall be in the range of 0 to $(\text{PicWidthInSamples}_L / \text{CropUnitX}) - (\text{frame_crop_right_offset} + 1)$, inclusive; and the value of **frame_crop_top_offset** shall be in the range of 0 to $(16 * \text{FrameHeightInMbs} / \text{CropUnitY}) - (\text{frame_crop_bottom_offset} + 1)$, inclusive.

When **frame_cropping_flag** is equal to 0, the values of **frame_crop_left_offset**, **frame_crop_right_offset**, **frame_crop_top_offset**, and **frame_crop_bottom_offset** shall be inferred to be equal to 0.

When ChromaArrayType is not equal to 0, the corresponding specified samples of the two chroma arrays are the samples having frame coordinates $(x / \text{SubWidthC}, y / \text{SubHeightC})$, where (x, y) are the frame coordinates of the specified luma samples.

For decoded fields, the specified samples of the decoded field are the samples that fall within the rectangle specified in frame coordinates.

vui_parameters_present_flag equal to 1 specifies that the **vui_parameters()** syntax structure as specified in Annex E is present. **vui_parameters_present_flag** equal to 0 specifies that the **vui_parameters()** syntax structure as specified in Annex E is not present.

7.4.2.1.1.1 Scaling list semantics

delta_scale is used to derive the j -th element of the scaling list for j in the range of 0 to $\text{sizeOfScalingList} - 1$, inclusive. The value of **delta_scale** shall be in the range of -128 to +127, inclusive.

When **useDefaultScalingMatrixFlag** is derived to be equal to 1, the scaling list shall be inferred to be equal to the default scaling list as specified in Table 7-2.

7.4.2.1.1.2 Sequence parameter set extension RBSP semantics

seq_parameter_set_id identifies the sequence parameter set associated with the sequence parameter set extension. The value of **seq_parameter_set_id** shall be in the range of 0 to 31, inclusive.

aux_format_idc equal to 0 indicates that there are no auxiliary coded pictures in the coded video sequence. **aux_format_idc** equal to 1 indicates that exactly one auxiliary coded picture is present in each access unit of the coded video sequence, and that for alpha blending purposes the decoded samples of the associated primary coded picture in

each access unit should be multiplied by the interpretation sample values of the auxiliary coded picture in the access unit in the display process after output from the decoding process. `aux_format_idc` equal to 2 indicates that exactly one auxiliary coded picture exists in each access unit of the coded video sequence, and that for alpha blending purposes the decoded samples of the associated primary coded picture in each access unit should not be multiplied by the interpretation sample values of the auxiliary coded picture in the access unit in the display process after output from the decoding process. `aux_format_idc` equal to 3 indicates that exactly one auxiliary coded picture exists in each access unit of the coded video sequence, and that the usage of the auxiliary coded pictures is unspecified. The value of `aux_format_idc` shall be in the range of 0 to 3, inclusive. Values greater than 3 for `aux_format_idc` are reserved to indicate the presence of exactly one auxiliary coded picture in each access unit of the coded video sequence for purposes to be specified in the future by ITU-T | ISO/IEC. When `aux_format_idc` is not present, it shall be inferred to be equal to 0.

NOTE 1 – Decoders conforming to this Recommendation | International Standard are not required to decode auxiliary coded pictures.

`bit_depth_aux_minus8` specifies the bit depth of the samples of the sample array of the auxiliary coded picture. `bit_depth_aux_minus8` shall be in the range of 0 to 4, inclusive.

`alpha_incr_flag` equal to 0 indicates that the interpretation sample value for each decoded auxiliary coded picture sample value is equal to the decoded auxiliary coded picture sample value for purposes of alpha blending. `alpha_incr_flag` equal to 1 indicates that, for purposes of alpha blending, after decoding the auxiliary coded picture samples, any auxiliary coded picture sample value that is greater than $\text{Min}(\text{alpha_opaque_value}, \text{alpha_transparent_value})$ should be increased by one to obtain the interpretation sample value for the auxiliary coded picture sample, and any auxiliary coded picture sample value that is less than or equal to $\text{Min}(\text{alpha_opaque_value}, \text{alpha_transparent_value})$ should be used without alteration as the interpretation sample value for the decoded auxiliary coded picture sample value.

`alpha_opaque_value` specifies the interpretation sample value of an auxiliary coded picture sample for which the associated luma and chroma samples of the same access unit are considered opaque for purposes of alpha blending. The number of bits used for the representation of the `alpha_opaque_value` syntax element is `bit_depth_aux_minus8` + 9 bits.

`alpha_transparent_value` specifies the interpretation sample value of an auxiliary coded picture sample for which the associated luma and chroma samples of the same access unit are considered transparent for purposes of alpha blending. The number of bits used for the representation of the `alpha_transparent_value` syntax element is `bit_depth_aux_minus8` + 9 bits.

When `alpha_incr_flag` is equal to 1, `alpha_transparent_value` shall not be equal to `alpha_opaque_value` and $\text{Log}_2(\text{Abs}(\text{alpha_opaque_value} - \text{alpha_transparent_value}))$ shall have an integer value. A value of `alpha_transparent_value` that is equal to `alpha_opaque_value` indicates that the auxiliary coded picture is not intended for alpha blending purposes.

NOTE 2 – For alpha blending purposes, `alpha_opaque_value` may be greater than `alpha_transparent_value`, or it may be less than `alpha_transparent_value`. Interpretation sample values should be clipped to the range of `alpha_opaque_value` to `alpha_transparent_value`, inclusive.

The decoding of the sequence parameter set extension and the decoding of auxiliary coded pictures is not required for conformance with this Recommendation | International Standard.

The syntax of each coded slice of an auxiliary coded picture shall obey the same constraints as a coded slice of a redundant picture, with the following differences of constraints.

- The following applies in regard to whether the primary coded picture is an IDR picture:
 - If the primary coded picture is an IDR picture, the auxiliary coded slice syntax shall correspond to that of a slice having `nal_unit_type` equal to 5 (a slice of an IDR picture);
 - Otherwise (the primary coded picture is not an IDR picture), the auxiliary coded slice syntax shall correspond to that of a slice having `nal_unit_type` equal to 1 (a slice of a non-IDR picture).
- The slices of an auxiliary coded picture (when present) shall contain all macroblocks corresponding to those of the primary coded picture.
- `redundant_pic_cnt` shall be equal to 0 in all auxiliary coded slices.

The (optional) decoding process for the decoding of auxiliary coded pictures is the same as if the auxiliary coded pictures were primary coded pictures in a separate coded video stream that differs from the primary coded pictures in the current coded video stream in the following ways.

- The IDR or non-IDR status of each auxiliary coded picture shall be inferred to be the same as the IDR or non-IDR status of the primary picture in the same access unit, rather than being inferred from the value of `nal_ref_idc`.

- The value of `chroma_format_idc` and the value of `ChromaArrayType` shall be inferred to be equal to 0 for the decoding of the auxiliary coded pictures.
- The value of `bit_depth_luma_minus8` shall be inferred to be equal to `bit_depth_aux_minus8` for the decoding of the auxiliary coded pictures.

NOTE 3 – Alpha blending composition is normally performed with a background picture B, a foreground picture F, and a decoded auxiliary coded picture A, all of the same size. Assume for purposes of example illustration that the chroma resolution of B and F have been upsampled to the same resolution as the luma. Denote corresponding samples of B, F and A by b , f and a , respectively. Denote luma and chroma samples by subscripts Y, Cb and Cr.

Define the variables `alphaRange`, `alphaFwt` and `alphaBwt` as follows:

$\text{alphaRange} = \text{Abs}(\text{alpha_opaque_value} - \text{alpha_transparent_value})$

$\text{alphaFwt} = \text{Abs}(a - \text{alpha_transparent_value})$

$\text{alphaBwt} = \text{Abs}(a - \text{alpha_opaque_value})$

Then, in alpha blending composition, samples d of the displayed picture D may be calculated as

$d_Y = (\text{alphaFwt} \cdot f_Y + \text{alphaBwt} \cdot b_Y + \text{alphaRange}/2) / \text{alphaRange}$

$d_{CB} = (\text{alphaFwt} \cdot f_{CB} + \text{alphaBwt} \cdot b_{CB} + \text{alphaRange}/2) / \text{alphaRange}$

$d_{CR} = (\text{alphaFwt} \cdot f_{CR} + \text{alphaBwt} \cdot b_{CR} + \text{alphaRange}/2) / \text{alphaRange}$

The samples of pictures D, F and B could also represent red, green, and blue component values (see subclause E.2.1). Here we have assumed Y, Cb and Cr component values. Each component, e.g., Y, is assumed for purposes of example illustration above to have the same bit depth in each of the pictures D, F and B. However, different components, e.g., Y and Cb, need not have the same bit depth in this example.

When `aux_format_idc` is equal to 1, F would be the decoded picture obtained from the decoded luma and chroma, and A would be the decoded picture obtained from the decoded auxiliary coded picture. In this case, the indicated example alpha blending composition involves multiplying the samples of F by factors obtained from the samples of A.

A picture format that is useful for editing or direct viewing, and that is commonly used, is called pre-multiplied-black video. If the foreground picture was F, then the pre-multiplied-black video S is given by

$s_Y = (\text{alphaFwt} \cdot f_Y) / \text{alphaRange}$

$s_{CB} = (\text{alphaFwt} \cdot f_{CB}) / \text{alphaRange}$

$s_{CR} = (\text{alphaFwt} \cdot f_{CR}) / \text{alphaRange}$

Pre-multiplied-black video has the characteristic that the picture S will appear correct if displayed against a black background. For a non-black background B, the composition of the displayed picture D may be calculated as

$d_Y = s_Y + (\text{alphaBwt} \cdot b_Y + \text{alphaRange}/2) / \text{alphaRange}$

$d_{CB} = s_{CB} + (\text{alphaBwt} \cdot b_{CB} + \text{alphaRange}/2) / \text{alphaRange}$

$d_{CR} = s_{CR} + (\text{alphaBwt} \cdot b_{CR} + \text{alphaRange}/2) / \text{alphaRange}$

When `aux_format_idc` is equal to 2, S would be the decoded picture obtained from the decoded luma and chroma, and A would again be the decoded picture obtained from the decoded auxiliary coded picture. In this case, alpha blending composition does not involve multiplication of the samples of S by factors obtained from the samples of A.

additional_extension_flag equal to 0 indicates that no additional data follows within the sequence parameter set extension syntax structure prior to the RBSP trailing bits. The value of `additional_extension_flag` shall be equal to 0. The value of 1 for `additional_extension_flag` is reserved for future use by ITU-T | ISO/IEC. Decoders that conform to this Recommendation | International Standard shall ignore all data that follows the value of 1 for `additional_extension_flag` in a sequence parameter set extension NAL unit.

7.4.2.2 Picture parameter set RBSP semantics

pic_parameter_set_id identifies the picture parameter set that is referred to in the slice header. The value of `pic_parameter_set_id` shall be in the range of 0 to 255, inclusive.

seq_parameter_set_id refers to the active sequence parameter set. The value of `seq_parameter_set_id` shall be in the range of 0 to 31, inclusive.

entropy_coding_mode_flag selects the entropy decoding method to be applied for the syntax elements for which two descriptors appear in the syntax tables as follows.

- If `entropy_coding_mode_flag` is equal to 0, the method specified by the left descriptor in the syntax table is applied (Exp-Golomb coded, see subclause 9.1 or CAVLC, see subclause 9.2).
- Otherwise (`entropy_coding_mode_flag` is equal to 1), the method specified by the right descriptor in the syntax table is applied (CABAC, see subclause 9.3).

pic_order_present_flag equal to 1 specifies that the picture order count related syntax elements are present in the slice headers as specified in subclause 7.3.3. `pic_order_present_flag` equal to 0 specifies that the picture order count related syntax elements are not present in the slice headers.

num_slice_groups_minus1 plus 1 specifies the number of slice groups for a picture. When **num_slice_groups_minus1** is equal to 0, all slices of the picture belong to the same slice group. The allowed range of **num_slice_groups_minus1** is specified in Annex A.

slice_group_map_type specifies how the mapping of slice group map units to slice groups is coded. The value of **slice_group_map_type** shall be in the range of 0 to 6, inclusive.

slice_group_map_type equal to 0 specifies interleaved slice groups.

slice_group_map_type equal to 1 specifies a dispersed slice group mapping.

slice_group_map_type equal to 2 specifies one or more “foreground” slice groups and a “leftover” slice group.

slice_group_map_type values equal to 3, 4, and 5 specify changing slice groups. When **num_slice_groups_minus1** is not equal to 1, **slice_group_map_type** shall not be equal to 3, 4, or 5.

slice_group_map_type equal to 6 specifies an explicit assignment of a slice group to each slice group map unit.

Slice group map units are specified as follows.

- If **frame_mbs_only_flag** is equal to 0 and **mb_adaptive_frame_field_flag** is equal to 1 and the coded picture is a frame, the slice group map units are macroblock pair units.
- Otherwise, if **frame_mbs_only_flag** is equal to 1 or a coded picture is a field, the slice group map units are units of macroblocks.
- Otherwise (**frame_mbs_only_flag** is equal to 0 and **mb_adaptive_frame_field_flag** is equal to 0 and the coded picture is a frame), the slice group map units are units of two macroblocks that are vertically contiguous as in a frame macroblock pair of an MBAFF frame.

run_length_minus1[i] is used to specify the number of consecutive slice group map units to be assigned to the *i*-th slice group in raster scan order of slice group map units. The value of **run_length_minus1[i]** shall be in the range of 0 to **PicSizeInMapUnits - 1**, inclusive.

top_left[i] and **bottom_right[i]** specify the top-left and bottom-right corners of a rectangle, respectively. **top_left[i]** and **bottom_right[i]** are slice group map unit positions in a raster scan of the picture for the slice group map units. For each rectangle *i*, all of the following constraints shall be obeyed by the values of the syntax elements **top_left[i]** and **bottom_right[i]**

- **top_left[i]** shall be less than or equal to **bottom_right[i]** and **bottom_right[i]** shall be less than **PicSizeInMapUnits**.
- $(\text{top_left}[i] \% \text{PicWidthInMbs})$ shall be less than or equal to the value of $(\text{bottom_right}[i] \% \text{PicWidthInMbs})$.

slice_group_change_direction_flag is used with **slice_group_map_type** to specify the refined map type when **slice_group_map_type** is 3, 4, or 5.

slice_group_change_rate_minus1 is used to specify the variable **SliceGroupChangeRate**. **SliceGroupChangeRate** specifies the multiple in number of slice group map units by which the size of a slice group can change from one picture to the next. The value of **slice_group_change_rate_minus1** shall be in the range of 0 to **PicSizeInMapUnits - 1**, inclusive. The **SliceGroupChangeRate** variable is specified as follows:

$$\text{SliceGroupChangeRate} = \text{slice_group_change_rate_minus1} + 1 \quad (7-21)$$

pic_size_in_map_units_minus1 is used to specify the number of slice group map units in the picture. **pic_size_in_map_units_minus1** shall be equal to **PicSizeInMapUnits - 1**.

slice_group_id[i] identifies a slice group of the *i*-th slice group map unit in raster scan order. The size of the **slice_group_id[i]** syntax element is $\text{Ceil}(\text{Log2}(\text{num_slice_groups_minus1} + 1))$ bits. The value of **slice_group_id[i]** shall be in the range of 0 to **num_slice_groups_minus1**, inclusive.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode each slice of the picture in which list 0 prediction is used when **num_ref_idx_active_override_flag** is equal to 0 for the slice. When **MbaffFrameFlag** is equal to 1, **num_ref_idx_l0_active_minus1** is the maximum index value for the decoding of frame macroblocks and $2 * \text{num_ref_idx_l0_active_minus1} + 1$ is the maximum index value for the decoding of field macroblocks. The value of **num_ref_idx_l0_active_minus1** shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as **num_ref_idx_l0_active_minus1** with l0 and list 0 replaced by l1 and list 1, respectively.

weighted_pred_flag equal to 0 specifies that weighted prediction shall not be applied to P and SP slices. **weighted_pred_flag** equal to 1 specifies that weighted prediction shall be applied to P and SP slices.

weighted_bipred_idc equal to 0 specifies that the default weighted prediction shall be applied to B slices. **weighted_bipred_idc** equal to 1 specifies that explicit weighted prediction shall be applied to B slices. **weighted_bipred_idc** equal to 2 specifies that implicit weighted prediction shall be applied to B slices. The value of **weighted_bipred_idc** shall be in the range of 0 to 2, inclusive.

pic_init_qp_minus26 specifies the initial value minus 26 of SliceQP_Y for each slice. The initial value is modified at the slice layer when a non-zero value of **slice_qp_delta** is decoded, and is modified further when a non-zero value of **mb_qp_delta** is decoded at the macroblock layer. The value of **pic_init_qp_minus26** shall be in the range of $-(26 + \text{QpBdOffset}_Y)$ to +25, inclusive.

pic_init_qs_minus26 specifies the initial value minus 26 of SliceQS_Y for all macroblocks in SP or SI slices. The initial value is modified at the slice layer when a non-zero value of **slice_qs_delta** is decoded. The value of **pic_init_qs_minus26** shall be in the range of -26 to +25, inclusive.

chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cb chroma component. The value of **chroma_qp_index_offset** shall be in the range of -12 to +12, inclusive.

deblocking_filter_control_present_flag equal to 1 specifies that a set of syntax elements controlling the characteristics of the deblocking filter is present in the slice header. **deblocking_filter_control_present_flag** equal to 0 specifies that the set of syntax elements controlling the characteristics of the deblocking filter is not present in the slice headers and their inferred values are in effect.

constrained_intra_pred_flag equal to 0 specifies that intra prediction allows usage of residual data and decoded samples of neighbouring macroblocks coded using Inter macroblock prediction modes for the prediction of macroblocks coded using Intra macroblock prediction modes. **constrained_intra_pred_flag** equal to 1 specifies constrained intra prediction, in which case prediction of macroblocks coded using Intra macroblock prediction modes only uses residual data and decoded samples from I or SI macroblock types.

redundant_pic_cnt_present_flag equal to 0 specifies that the **redundant_pic_cnt** syntax element is not present in slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set. **redundant_pic_cnt_present_flag** equal to 1 specifies that the **redundant_pic_cnt** syntax element is present in all slice headers, data partitions B, and data partitions C that refer (either directly or by association with a corresponding data partition A) to the picture parameter set.

transform_8x8_mode_flag equal to 1 specifies that the 8x8 transform decoding process may be in use (see subclause 8.5). **transform_8x8_mode_flag** equal to 0 specifies that the 8x8 transform decoding process is not in use. When **transform_8x8_mode_flag** is not present, it shall be inferred to be 0.

pic_scaling_matrix_present_flag equal to 1 specifies that parameters are present to modify the scaling lists specified in the sequence parameter set. **pic_scaling_matrix_present_flag** equal to 0 specifies that the scaling lists used for the picture shall be inferred to be equal to those specified by the sequence parameter set. When **pic_scaling_matrix_present_flag** is not present, it shall be inferred to be equal to 0.

pic_scaling_list_present_flag[i] equal to 1 specifies that the scaling list syntax structure is present to specify the scaling list for index i. **pic_scaling_list_present_flag[i]** equal to 0 specifies that the syntax structure for scaling list i is not present in the picture parameter set and that depending on the value of **seq_scaling_matrix_present_flag**, the following applies.

- If **seq_scaling_matrix_present_flag** is equal to 0, the scaling list fall-back rule set A as specified in Table 7-2 shall be used to derive the picture-level scaling list for index i.
- Otherwise (**seq_scaling_matrix_present_flag** is equal to 1), the scaling list fall-back rule set B as specified in Table 7-2 shall be used to derive the picture-level scaling list for index i.

second_chroma_qp_index_offset specifies the offset that shall be added to QP_Y and QS_Y for addressing the table of QP_C values for the Cr chroma component. The value of **second_chroma_qp_index_offset** shall be in the range of -12 to +12, inclusive.

When **second_chroma_qp_index_offset** is not present, it shall be inferred to be equal to **chroma_qp_index_offset**.

NOTE – When **ChromaArrayType** is equal to 0, the values of **bit_depth_chroma_minus8**, **chroma_qp_index_offset** and **second_chroma_qp_index_offset** are not used in the decoding process. In particular, when **separate_colour_plane_flag** is equal to 1, each colour plane is decoded as a distinct monochrome picture using the luma component decoding process (except for the selection of scaling matrices), including the application of the luma quantisation parameter derivation process without application of an offset for the decoding of the pictures having **colour_plane_id** not equal to 0.

7.4.2.3 Supplemental enhancement information RBSP semantics

Supplemental Enhancement Information (SEI) contains information that is not necessary to decode the samples of coded pictures from VCL NAL units.

7.4.2.3.1 Supplemental enhancement information message semantics

An SEI NAL unit contains one or more SEI messages. Each SEI message consists of the variables specifying the type payloadType and size payloadSize of the SEI payload. SEI payloads are specified in Annex D. The derived SEI payload size payloadSize is specified in bytes and shall be equal to the number of bytes in the SEI payload.

ff_byte is a byte equal to 0xFF identifying a need for a longer representation of the syntax structure that it is used within.

last_payload_type_byte is the last byte of the payload type of an SEI message.

last_payload_size_byte is the last byte of the size of an SEI message.

7.4.2.4 Access unit delimiter RBSP semantics

The access unit delimiter may be used to indicate the type of slices present in a primary coded picture and to simplify the detection of the boundary between access units. There is no normative decoding process associated with the access unit delimiter.

primary_pic_type indicates that the slice_type values for all slices of the primary coded picture are members of the set listed in Table 7-5 for the given value of primary_pic_type.

Table 7-5 – Meaning of primary_pic_type

primary_pic_type	slice_type values that may be present in the primary coded picture
0	2, 7
1	0, 2, 5, 7
2	0, 1, 2, 5, 6, 7
3	4, 9
4	3, 4, 8, 9
5	2, 4, 7, 9
6	0, 2, 3, 4, 5, 7, 8, 9
7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

7.4.2.5 End of sequence RBSP semantics

The end of sequence RBSP specifies that the next subsequent access unit in the bitstream in decoding order (if any) shall be an IDR access unit. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty. No normative decoding process is specified for an end of sequence RBSP.

7.4.2.6 End of stream RBSP semantics

The end of stream RBSP indicates that no additional NAL units shall be present in the bitstream that are subsequent to the end of stream RBSP in decoding order. The syntax content of the SODB and RBSP for the end of stream RBSP are empty. No normative decoding process is specified for an end of stream RBSP.

7.4.2.7 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF. No normative decoding process is specified for a filler data RBSP.

ff_byte is a byte equal to 0xFF.

7.4.2.8 Slice layer without partitioning RBSP semantics

The slice layer without partitioning RBSP consists of a slice header and slice data.

7.4.2.9 Slice data partition RBSP semantics

7.4.2.9.1 Slice data partition A RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Partition A contains all syntax elements of category 2.

Category 2 syntax elements include all syntax elements in the slice header and slice data syntax structures other than the syntax elements in the residual() syntax structure.

slice_id identifies the slice associated with the data partition. The value of **slice_id** is constrained as follows.

- If **separate_colour_plane_flag** is equal to 0, the following applies.
 - If arbitrary slice order is not allowed as specified in Annex A, the first slice of a coded picture, in decoding order, shall have **slice_id** equal to 0 and the value of **slice_id** shall be incremented by one for each subsequent slice of the coded picture in decoding order.
 - Otherwise (arbitrary slice order is allowed), each slice shall have a unique **slice_id** value within the set of slices of the coded picture.
- Otherwise (**separate_colour_plane_flag** is equal to 1), the following applies.
 - If arbitrary slice order is not allowed as specified in Annex A, the first slice of a coded picture having each value of **colour_plane_id**, in decoding order, shall have **slice_id** equal to 0 and the value of **slice_id** shall be incremented by one for each subsequent slice of the coded picture having the same value of **colour_plane_id**, in decoding order.
 - Otherwise (arbitrary slice order is allowed) each slice shall have a unique **slice_id** value within each set of slices of the coded picture that have the same value of **colour_plane_id**.

The range of **slice_id** is specified as follows.

- If **MbaffFrameFlag** is equal to 0, **slice_id** shall be in the range of 0 to **PicSizeInMbs** - 1, inclusive.
- Otherwise (**MbaffFrameFlag** is equal to 1), **slice_id** shall be in the range of 0 to **PicSizeInMbs** / 2 - 1, inclusive.

7.4.2.9.2 Slice data partition B RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into one to three separate partitions. Slice data partition B contains all syntax elements of category 3.

Category 3 syntax elements include all syntax elements in the residual() syntax structure and in syntax structures used within that syntax structure for collective macroblock types I and SI as specified in Table 7-10.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

colour_plane_id specifies the colour plane associated with the current slice RBSP when **separate_colour_plane_flag** is equal to 1. The value of **colour_plane_id** shall be in the range of 0 to 2, inclusive. **colour_plane_id** equal to 0, 1, and 2 correspond to the Y, Cb, and Cr planes, respectively.

NOTE – There is no dependency between the decoding processes of pictures having different values of **colour_plane_id**.

redundant_pic_cnt shall be equal to 0 for slices and slice data partitions belonging to the primary coded picture. The **redundant_pic_cnt** shall be greater than 0 for coded slices and coded slice data partitions in redundant coded pictures. When **redundant_pic_cnt** is not present, its value shall be inferred to be equal to 0. The value of **redundant_pic_cnt** shall be in the range of 0 to 127, inclusive.

The presence of a slice data partition B RBSP is specified as follows.

- If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 3 in the slice data for a slice, a slice data partition B RBSP shall be present having the same value of **slice_id** and **redundant_pic_cnt** as in the slice data partition A RBSP.
- Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 3 in the slice data for a slice), no slice data partition B RBSP shall be present having the same value of **slice_id** and **redundant_pic_cnt** as in the slice data partition A RBSP.

7.4.2.9.3 Slice data partition C RBSP semantics

When slice data partitioning is in use, the coded data for a single slice is divided into three separate partitions. Slice data partition C contains all syntax elements of category 4.

Category 4 syntax elements include all syntax elements in the residual() syntax structure and in syntax structures used within that syntax structure for collective macroblock types P and B as specified in Table 7-10.

slice_id has the same semantics as specified in subclause 7.4.2.9.1.

colour_plane_id has the same semantics as specified in subclause 7.4.2.9.2.

redundant_pic_cnt has the same semantics as specified in subclause 7.4.2.9.2.

The presence of a slice data partition C RBSP is specified as follows.

- If the syntax elements of a slice data partition A RBSP indicate the presence of any syntax elements of category 4 in the slice data for a slice, a slice data partition C RBSP shall be present having the same value of **slice_id** and **redundant_pic_cnt** as in the slice data partition A RBSP.
- Otherwise (the syntax elements of a slice data partition A RBSP do not indicate the presence of any syntax elements of category 4 in the slice data for a slice), no slice data partition C RBSP shall be present having the same value of **slice_id** and **redundant_pic_cnt** as in the slice data partition A RBSP.

7.4.2.10 RBSP slice trailing bits semantics

cabac_zero_word is a byte-aligned sequence of two bytes equal to 0x0000.

Let **NumBytesInVclNALunits** be the sum of the values of **NumBytesInNALunit** for all VCL NAL units of a coded picture.

Let **BinCountsInNALunits** be the number of times that the parsing process function **DecodeBin()**, specified in subclause 9.3.3.2, is invoked to decode the contents of all VCL NAL units of a coded picture. When **entropy_coding_mode_flag** is equal to 1, **BinCountsInNALunits** shall not exceed $(32 \div 3) * \text{NumBytesInVclNALunits} + (\text{RawMbBits} * \text{PicSizeInMbs}) \div 32$.

NOTE – The constraint on the maximum number of bins resulting from decoding the contents of the slice layer NAL units can be met by inserting a number of **cabac_zero_word** syntax elements to increase the value of **NumBytesInVclNALunits**. Each **cabac_zero_word** is represented in a NAL unit by the three-byte sequence 0x000003 (as a result of the constraints on NAL unit contents that result in requiring inclusion of an **emulation_prevention_three_byte** for each **cabac_zero_word**).

7.4.2.11 RBSP trailing bits semantics

rbbsp_stop_one_bit shall be equal to 1.

rbbsp_alignment_zero_bit shall be equal to 0.

7.4.3 Slice header semantics

When present, the value of the slice header syntax elements **pic_parameter_set_id**, **frame_num**, **field_pic_flag**, **bottom_field_flag**, **idr_pic_id**, **pic_order_cnt_lsb**, **delta_pic_order_cnt_bottom**, **delta_pic_order_cnt[0]**, **delta_pic_order_cnt[1]**, **sp_for_switch_flag**, and **slice_group_change_cycle** shall be the same in all slice headers of a coded picture.

first_mb_in_slice specifies the address of the first macroblock in the slice. When arbitrary slice order is not allowed as specified in Annex A, the value of **first_mb_in_slice** is constrained as follows.

- If **separate_colour_plane_flag** is equal to 0, the value of **first_mb_in_slice** shall not be less than the value of **first_mb_in_slice** for any other slice of the current picture that precedes the current slice in decoding order.
- Otherwise (**separate_colour_plane_flag** is equal to 1), the value of **first_mb_in_slice** shall not be less than the value of **first_mb_in_slice** for any other slice of the current picture that precedes the current slice in decoding order and has the same value of **colour_plane_id**.

The first macroblock address of the slice is derived as follows.

- If **MbaffFrameFlag** is equal to 0, **first_mb_in_slice** is the macroblock address of the first macroblock in the slice, and **first_mb_in_slice** shall be in the range of 0 to **PicSizeInMbs** – 1, inclusive.
- Otherwise (**MbaffFrameFlag** is equal to 1), **first_mb_in_slice * 2** is the macroblock address of the first macroblock in the slice, which is the top macroblock of the first macroblock pair in the slice, and **first_mb_in_slice** shall be in the range of 0 to **PicSizeInMbs** / 2 – 1, inclusive.

slice_type specifies the coding type of the slice according to Table 7-6.

Table 7-6 – Name association to slice_type

slice_type	Name of slice_type
0	P (P slice)
1	B (B slice)
2	I (I slice)
3	SP (SP slice)
4	SI (SI slice)
5	P (P slice)
6	B (B slice)
7	I (I slice)
8	SP (SP slice)
9	SI (SI slice)

slice_type values in the range 5..9 specify, in addition to the coding type of the current slice, that all other slices of the current coded picture shall have a value of slice_type equal to the current value of slice_type or equal to the current value of slice_type – 5.

When nal_unit_type is equal to 5 (IDR picture), slice_type shall be equal to 2, 4, 7, or 9.

When num_ref_frames is equal to 0, slice_type shall be equal to 2, 4, 7, or 9.

pic_parameter_set_id specifies the picture parameter set in use. The value of pic_parameter_set_id shall be in the range of 0 to 255, inclusive.

colour_plane_id specifies the colour plane associated with the current slice RBSP when separate_colour_plane_flag is equal to 1. The value of colour_plane_id shall be in the range of 0 to 2, inclusive. colour_plane_id equal to 0, 1, and 2 correspond to the Y, Cb, and Cr planes, respectively.

NOTE 1 – There is no dependency between the decoding processes of pictures having different values of colour_plane_id.

frame_num is used as an identifier for pictures and shall be represented by $\log_2 \text{max_frame_num} - 4 + 4$ bits in the bitstream. frame_num is constrained as follows:

The variable PrevRefFrameNum is derived as follows.

- If the current picture is an IDR picture, PrevRefFrameNum is set equal to 0.
- Otherwise (the current picture is not an IDR picture), PrevRefFrameNum is set as follows.
 - If the decoding process for gaps in frame_num specified in subclause 8.2.5.2 was invoked by the decoding process for an access unit that contained a non-reference picture that followed the previous access unit in decoding order that contained a reference picture, PrevRefFrameNum is set equal to the value of frame_num for the last of the "non-existing" reference frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.
 - Otherwise, PrevRefFrameNum is set equal to the value of frame_num for the previous access unit in decoding order that contained a reference picture.

The value of frame_num is constrained as follows.

- If the current picture is an IDR picture, frame_num shall be equal to 0.
- Otherwise (the current picture is not an IDR picture), referring to the primary coded picture in the previous access unit in decoding order that contains a reference picture as the preceding reference picture, the value of frame_num for the current picture shall not be equal to PrevRefFrameNum unless all of the following three conditions are true.
 - the current picture and the preceding reference picture belong to consecutive access units in decoding order
 - the current picture and the preceding reference picture are reference fields having opposite parity
 - one or more of the following conditions is true
 - the preceding reference picture is an IDR picture
 - the preceding reference picture includes a memory_management_control_operation syntax element equal to 5

NOTE 2 – When the preceding reference picture includes a `memory_management_control_operation` syntax element equal to 5, `PrevRefFrameNum` is equal to 0.

- there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture does not have `frame_num` equal to `PrevRefFrameNum`
- there is a primary coded picture that precedes the preceding reference picture and the primary coded picture that precedes the preceding reference picture is not a reference picture

When the value of `frame_num` is not equal to `PrevRefFrameNum`, the following applies.

- There shall not be any previous field or frame in decoding order that is currently marked as "used for short-term reference" that has a value of `frame_num` equal to any value taken on by the variable `UnusedShortTermFrameNum` in the following:

```
UnusedShortTermFrameNum = ( PrevRefFrameNum + 1 ) % MaxFrameNum
while( UnusedShortTermFrameNum != frame_num )
    UnusedShortTermFrameNum = ( UnusedShortTermFrameNum + 1 ) % MaxFrameNum
```

(7-22)

- The value of `frame_num` is constrained as follows.
 - If `gaps_in_frame_num_value_allowed_flag` is equal to 0, the value of `frame_num` for the current picture shall be equal to $(\text{PrevRefFrameNum} + 1) \% \text{MaxFrameNum}$.
 - Otherwise (`gaps_in_frame_num_value_allowed_flag` is equal to 1), the following applies.
 - If `frame_num` is greater than `PrevRefFrameNum`, there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which either of the following conditions is true.
 - The value of `frame_num` for the non-reference picture is less than `PrevRefFrameNum`.
 - The value of `frame_num` for the non-reference picture is greater than the value of `frame_num` for the current picture.
 - Otherwise (`frame_num` is less than `PrevRefFrameNum`), there shall not be any non-reference pictures in the bitstream that follow the previous reference picture and precede the current picture in decoding order in which both of the following conditions are true.
 - The value of `frame_num` for the non-reference picture is less than `PrevRefFrameNum`.
 - The value of `frame_num` for the non-reference picture is greater than the value of `frame_num` for the current picture.

A picture including a `memory_management_control_operation` equal to 5 shall have `frame_num` constraints as described above and, after the decoding of the current picture and the processing of the memory management control operations, the picture shall be inferred to have had `frame_num` equal to 0 for all subsequent use in the decoding process, except as specified in subclause 7.4.1.2.4.

NOTE 3 – When the primary coded picture is not an IDR picture and does not contain `memory_management_control_operation` syntax element equal to 5, the value of `frame_num` of a corresponding redundant coded picture is the same as the value of `frame_num` in the primary coded picture. Alternatively, the redundant coded picture includes a `memory_management_control_operation` syntax element equal to 5 and the corresponding primary coded picture is an IDR picture.

field_pic_flag equal to 1 specifies that the slice is a slice of a coded field. **field_pic_flag** equal to 0 specifies that the slice is a slice of a coded frame. When **field_pic_flag** is not present it shall be inferred to be equal to 0.

The variable `MbaffFrameFlag` is derived as follows

$$\text{MbaffFrameFlag} = (\text{mb_adaptive_frame_field_flag} \ \&\& \ !\text{field_pic_flag})$$

(7-23)

The variable for the picture height in units of macroblocks is derived as follows

$$\text{PicHeightInMbs} = \text{FrameHeightInMbs} / (1 + \text{field_pic_flag})$$

(7-24)

The variable for picture height for the luma component is derived as follows

$$\text{PicHeightInSamples}_L = \text{PicHeightInMbs} * 16$$

(7-25)

The variable for picture height for the chroma component is derived as follows

$$\text{PicHeightInSamples}_C = \text{PicHeightInMbs} * \text{MbHeightC} \quad (7-26)$$

The variable PicSizeInMbs for the current picture is derived according to:

$$\text{PicSizeInMbs} = \text{PicWidthInMbs} * \text{PicHeightInMbs} \quad (7-27)$$

The variable MaxPicNum is derived as follows.

- If field_pic_flag is equal to 0, MaxPicNum is set equal to MaxFrameNum .
- Otherwise (field_pic_flag is equal to 1), MaxPicNum is set equal to $2 * \text{MaxFrameNum}$.

The variable CurrPicNum is derived as follows.

- If field_pic_flag is equal to 0, CurrPicNum is set equal to frame_num .
- Otherwise (field_pic_flag is equal to 1), CurrPicNum is set equal to $2 * \text{frame_num} + 1$.

bottom_field_flag equal to 1 specifies that the slice is part of a coded bottom field, **bottom_field_flag** equal to 0 specifies that the picture is a coded top field. When this syntax element is not present for the current slice, it shall be inferred to be equal to 0.

idr_pic_id identifies an IDR picture. The values of idr_pic_id in all the slices of an IDR picture shall remain unchanged. When two consecutive access units in decoding order are both IDR access units, the value of idr_pic_id in the slices of the first such IDR access unit shall differ from the idr_pic_id in the second such IDR access unit. The value of idr_pic_id shall be in the range of 0 to 65535, inclusive.

pic_order_cnt_lsb specifies the picture order count modulo MaxPicOrderCntLsb for the top field of a coded frame or for a coded field. The size of the pic_order_cnt_lsb syntax element is $\log_2 \text{max_pic_order_cnt_lsb_minus4} + 4$ bits. The value of the pic_order_cnt_lsb shall be in the range of 0 to $\text{MaxPicOrderCntLsb} - 1$, inclusive.

delta_pic_order_cnt_bottom specifies the picture order count difference between the bottom field and the top field of a coded frame as follows.

- If the current picture includes a $\text{memory_management_control_operation}$ equal to 5, the value of $\text{delta_pic_order_cnt_bottom}$ shall be in the range of $(1 - \text{MaxPicOrderCntLsb})$ to $2^{31} - 1$, inclusive.
- Otherwise (the current picture does not include a $\text{memory_management_control_operation}$ equal to 5), the value of $\text{delta_pic_order_cnt_bottom}$ shall be in the range of -2^{31} to $2^{31} - 1$, inclusive.

When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

delta_pic_order_cnt[0] specifies the picture order count difference from the expected picture order count for the top field of a coded frame or for a coded field as specified in subclause 8.2.1. The value of $\text{delta_pic_order_cnt}[0]$ shall be in the range of -2^{31} to $2^{31} - 1$, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

delta_pic_order_cnt[1] specifies the picture order count difference from the expected picture order count for the bottom field of a coded frame specified in subclause 8.2.1. The value of $\text{delta_pic_order_cnt}[1]$ shall be in the range of -2^{31} to $2^{31} - 1$, inclusive. When this syntax element is not present in the bitstream for the current slice, it shall be inferred to be equal to 0.

redundant_pic_cnt shall be equal to 0 for slices and slice data partitions belonging to the primary coded picture. The value of redundant_pic_cnt shall be greater than 0 for coded slices or coded slice data partitions of a redundant coded picture. When redundant_pic_cnt is not present in the bitstream, its value shall be inferred to be equal to 0. The value of redundant_pic_cnt shall be in the range of 0 to 127, inclusive.

NOTE 4 – Any area of the decoded primary picture and the corresponding area that would result from application of the decoding process specified in clause 8 for any redundant picture in the same access unit should be visually similar in appearance.

The value of $\text{pic_parameter_set_id}$ in a coded slice or coded slice data partition of a redundant coded picture shall be such that the value of $\text{pic_order_present_flag}$ in the picture parameter set in use in a redundant coded picture is equal to the value of $\text{pic_order_present_flag}$ in the picture parameter set in use in the corresponding primary coded picture.

When present in the primary coded picture and any redundant coded picture, the following syntax elements shall have the same value: field_pic_flag , bottom_field_flag , and idr_pic_id .

When the value of `nal_ref_idc` in one VCL NAL unit of an access unit is equal to 0, the value of `nal_ref_idc` in all other VCL NAL units of the same access unit shall be equal to 0.

NOTE 5 – The above constraint also has the following implications. If the value of `nal_ref_idc` for the VCL NAL units of the primary coded picture is equal to 0, the value of `nal_ref_idc` for the VCL NAL units of any corresponding redundant coded picture are equal to 0; otherwise (the value of `nal_ref_idc` for the VCL NAL units of the primary coded picture is greater than 0), the value of `nal_ref_idc` for the VCL NAL units of any corresponding redundant coded picture are also greater than 0.

The marking status of reference pictures and the value of `frame_num` after the decoded reference picture marking process as specified in subclause 8.2.5 is invoked for the primary coded picture or any redundant coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

NOTE 6 – The above constraint also has the following implications.

If a primary coded picture is not an IDR picture, the contents of the `dec_ref_pic_marking()` syntax structure must be identical in all slice headers of the primary coded picture and all redundant coded pictures corresponding to the primary coded picture.

Otherwise (a primary coded picture is an IDR picture), the following applies.

If a redundant coded picture corresponding to the primary coded picture is an IDR picture, the contents of the `dec_ref_pic_marking()` syntax structure must be identical in all slice headers of the primary coded picture and the redundant coded picture corresponding to the primary coded picture.

Otherwise (a redundant picture corresponding to the primary coded picture is not an IDR picture), all slice headers of the redundant picture must contain a `dec_ref_pic_marking` syntax() structure including a `memory_management_control_operation` syntax element equal to 5, and the following applies.

If the value of `long_term_reference_flag` in the primary coded picture is equal to 0, the `dec_ref_pic_marking` syntax structure of the redundant coded picture must not include a `memory_management_control_operation` syntax element equal to 6.

Otherwise (the value of `long_term_reference_flag` in the primary coded picture is equal to 1), the `dec_ref_pic_marking` syntax structure of the redundant coded picture must include `memory_management_control_operation` syntax elements equal to 5, 4, and 6 in decoding order, and the value of `max_long_term_frame_idx_plus1` must be equal to 1, and the value of `long_term_frame_idx` must be equal to 0.

The values of `TopFieldOrderCnt` and `BottomFieldOrderCnt` (if applicable) that result after completion of the decoding process for any redundant coded picture or the primary coded picture of the same access unit shall be identical regardless whether the primary coded picture or any redundant coded picture (instead of the primary coded picture) of the access unit would be decoded.

There is no required decoding process for a coded slice or coded slice data partition of a redundant coded picture. When the `redundant_pic_cnt` in the slice header of a coded slice is greater than 0, the decoder may discard the coded slice. However, a coded slice or coded slice data partition of any redundant coded picture shall obey the same constraints as a coded slice or coded slice data partition of a primary picture.

NOTE 7 – When some of the samples in the decoded primary picture cannot be correctly decoded due to errors or losses in transmission of the sequence and a coded redundant slice can be correctly decoded, the decoder should replace the samples of the decoded primary picture with the corresponding samples of the decoded redundant slice. When more than one redundant slice covers the relevant region of the primary picture, the redundant slice having the lowest value of `redundant_pic_cnt` should be used.

Redundant slices and slice data partitions having the same value of `redundant_pic_cnt` belong to the same redundant picture. Decoded slices within the same redundant picture need not cover the entire picture area and shall not overlap.

`direct_spatial_mv_pred_flag` specifies the method used in the decoding process to derive motion vectors and reference indices for inter prediction as follows.

- If `direct_spatial_mv_pred_flag` is equal to 1, the derivation process for luma motion vectors for `B_Skip`, `B_Direct_16x16`, and `B_Direct_8x8` in subclause 8.4.1.2 shall use spatial direct mode prediction as specified in subclause 8.4.1.2.2.
- Otherwise (`direct_spatial_mv_pred_flag` is equal to 0), the derivation process for luma motion vectors for `B_Skip`, `B_Direct_16x16`, and `B_Direct_8x8` in subclause 8.4.1.2 shall use temporal direct mode prediction as specified in subclause 8.4.1.2.3.

`num_ref_idx_active_override_flag` equal to 0 specifies that the values of the syntax elements `num_ref_idx_l0_active_minus1` and `num_ref_idx_l1_active_minus1` specified in the referred picture parameter set are in effect. `num_ref_idx_active_override_flag` equal to 1 specifies that the `num_ref_idx_l0_active_minus1` and `num_ref_idx_l1_active_minus1` specified in the referred picture parameter set are overridden for the current slice (and only for the current slice) by the following values in the slice header.

When the current slice is a P, SP, or B slice and `field_pic_flag` is equal to 0 and the value of `num_ref_idx_l0_active_minus1` in the picture parameter set exceeds 15, `num_ref_idx_active_override_flag` shall be equal to 1.

When the current slice is a B slice and `field_pic_flag` is equal to 0 and the value of `num_ref_idx_l1_active_minus1` in the picture parameter set exceeds 15, `num_ref_idx_active_override_flag` shall be equal to 1.

num_ref_idx_l0_active_minus1 specifies the maximum reference index for reference picture list 0 that shall be used to decode the slice.

The range of `num_ref_idx_l0_active_minus1` is specified as follows.

- If `field_pic_flag` is equal to 0, `num_ref_idx_l0_active_minus1` shall be in the range of 0 to 15, inclusive. When `MbaffFrameFlag` is equal to 1, `num_ref_idx_l0_active_minus1` is the maximum index value for the decoding of frame macroblocks and $2 * \text{num_ref_idx_l0_active_minus1} + 1$ is the maximum index value for the decoding of field macroblocks.
- Otherwise (`field_pic_flag` is equal to 1), `num_ref_idx_l0_active_minus1` shall be in the range of 0 to 31, inclusive.

num_ref_idx_l1_active_minus1 has the same semantics as `num_ref_idx_l0_active_minus1` with l0 and list 0 replaced by l1 and list 1, respectively.

cabac_init_idc specifies the index for determining the initialisation table used in the initialisation process for context variables. The value of `cabac_init_idc` shall be in the range of 0 to 2, inclusive.

slice_qp_delta specifies the initial value of QP_Y to be used for all the macroblocks in the slice until modified by the value of `mb_qp_delta` in the macroblock layer. The initial QP_Y quantisation parameter for the slice is computed as:

$$\text{SliceQP}_Y = 26 + \text{pic_init_qp_minus26} + \text{slice_qp_delta} \quad (7-28)$$

The value of `slice_qp_delta` shall be limited such that SliceQP_Y is in the range of $-\text{QpBdOffset}_Y$ to +51, inclusive.

sp_for_switch_flag specifies the decoding process to be used to decode P macroblocks in an SP slice as follows.

- If `sp_for_switch_flag` is equal to 0, the P macroblocks in the SP slice shall be decoded using the SP decoding process for non-switching pictures as specified in subclause 8.6.1.
- Otherwise (`sp_for_switch_flag` is equal to 1), the P macroblocks in the SP slice shall be decoded using the SP and SI decoding process for switching pictures as specified in subclause 8.6.2.

slice_qs_delta specifies the value of QS_Y for all the macroblocks in SP and SI slices. The QS_Y quantisation parameter for the slice is computed as:

$$QS_Y = 26 + \text{pic_init_qs_minus26} + \text{slice_qs_delta} \quad (7-29)$$

The value of `slice_qs_delta` shall be limited such that QS_Y is in the range of 0 to 51, inclusive. This value of QS_Y is used for the decoding of all macroblocks in SI slices with `mb_type` equal to SI and all macroblocks in SP slices with prediction mode equal to inter.

disable_deblocking_filter_idc specifies whether the operation of the deblocking filter shall be disabled across some block edges of the slice and specifies for which edges the filtering is disabled. When `disable_deblocking_filter_idc` is not present in the slice header, the value of `disable_deblocking_filter_idc` shall be inferred to be equal to 0.

The value of `disable_deblocking_filter_idc` shall be in the range of 0 to 2, inclusive.

slice_alpha_c0_offset_div2 specifies the offset used in accessing the α and t_{c0} deblocking filter tables for filtering operations controlled by the macroblocks within the slice. From this value, the offset that shall be applied when addressing these tables shall be computed as:

$$\text{FilterOffsetA} = \text{slice_alpha_c0_offset_div2} \ll 1 \quad (7-30)$$

The value of `slice_alpha_c0_offset_div2` shall be in the range of -6 to +6, inclusive. When `slice_alpha_c0_offset_div2` is not present in the slice header, the value of `slice_alpha_c0_offset_div2` shall be inferred to be equal to 0.

slice_beta_offset_div2 specifies the offset used in accessing the β deblocking filter table for filtering operations controlled by the macroblocks within the slice. From this value, the offset that is applied when addressing the β table of the deblocking filter shall be computed as:

$$\text{FilterOffsetB} = \text{slice_beta_offset_div2} \ll 1 \quad (7-31)$$

The value of `slice_beta_offset_div2` shall be in the range of -6 to +6, inclusive. When `slice_beta_offset_div2` is not present in the slice header the value of `slice_beta_offset_div2` shall be inferred to be equal to 0.

`slice_group_change_cycle` is used to derive the number of slice group map units in slice group 0 when `slice_group_map_type` is equal to 3, 4, or 5, as specified by

$$\text{MapUnitsInSliceGroup0} = \text{Min}(\text{slice_group_change_cycle} * \text{SliceGroupChangeRate}, \text{PicSizeInMapUnits}) \quad (7-32)$$

The value of `slice_group_change_cycle` is represented in the bitstream by the following number of bits

$$\text{Ceil}(\text{Log2}(\text{PicSizeInMapUnits} + \text{SliceGroupChangeRate} + 1)) \quad (7-33)$$

The value of `slice_group_change_cycle` shall be in the range of 0 to $\text{Ceil}(\text{PicSizeInMapUnits} + \text{SliceGroupChangeRate})$, inclusive.

7.4.3.1 Reference picture list reordering semantics

The syntax elements `reordering_of_pic_nums_idc`, `abs_diff_pic_num_minus1`, and `long_term_pic_num` specify the change from the initial reference picture lists to the reference picture lists to be used for decoding the slice.

`ref_pic_list_reordering_flag_l0` equal to 1 specifies that the syntax element `reordering_of_pic_nums_idc` is present for specifying reference picture list 0. `ref_pic_list_reordering_flag_l0` equal to 0 specifies that this syntax element is not present.

When `ref_pic_list_reordering_flag_l0` is equal to 1, the number of times that `reordering_of_pic_nums_idc` is not equal to 3 following `ref_pic_list_reordering_flag_l0` shall not exceed `num_ref_idx_l0_active_minus1 + 1`.

When `RefPicList0[num_ref_idx_l0_active_minus1]` in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", `ref_pic_list_reordering_flag_l0` shall be equal to 1 and `reordering_of_pic_nums_idc` shall not be equal to 3 until `RefPicList0[num_ref_idx_l0_active_minus1]` in the reordered list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

`ref_pic_list_reordering_flag_l1` equal to 1 specifies that the syntax element `reordering_of_pic_nums_idc` is present for specifying reference picture list 1. `ref_pic_list_reordering_flag_l1` equal to 0 specifies that this syntax element is not present.

When `ref_pic_list_reordering_flag_l1` is equal to 1, the number of times that `reordering_of_pic_nums_idc` is not equal to 3 following `ref_pic_list_reordering_flag_l1` shall not exceed `num_ref_idx_l1_active_minus1 + 1`.

When decoding a slice with `slice_type` equal to 1 or 6 and `RefPicList1[num_ref_idx_l1_active_minus1]` in the initial reference picture list produced as specified in subclause 8.2.4.2 is equal to "no reference picture", `ref_pic_list_reordering_flag_l1` shall be equal to 1 and `reordering_of_pic_nums_idc` shall not be equal to 3 until `RefPicList1[num_ref_idx_l1_active_minus1]` in the reordered list produced as specified in subclause 8.2.4.3 is not equal to "no reference picture".

`reordering_of_pic_nums_idc` together with `abs_diff_pic_num_minus1` or `long_term_pic_num` specifies which of the reference pictures are re-mapped. The values of `reordering_of_pic_nums_idc` are specified in Table 7-7. The value of the first `reordering_of_pic_nums_idc` that follows immediately after `ref_pic_list_reordering_flag_l0` or `ref_pic_list_reordering_flag_l1` shall not be equal to 3.

Table 7-7 – `reordering_of_pic_nums_idc` operations for reordering of reference picture lists

<code>reordering_of_pic_nums_idc</code>	Reordering specified
0	<code>abs_diff_pic_num_minus1</code> is present and corresponds to a difference to subtract from a picture number prediction value
1	<code>abs_diff_pic_num_minus1</code> is present and corresponds to a difference to add to a picture number prediction value
2	<code>long_term_pic_num</code> is present and specifies the long-term picture number for a reference picture
3	End loop for reordering of the initial reference picture list

abs_diff_pic_num_minus1 plus 1 specifies the absolute difference between the picture number of the picture being moved to the current index in the list and the picture number prediction value. **abs_diff_pic_num_minus1** shall be in the range of 0 to **MaxPicNum** – 1. The allowed values of **abs_diff_pic_num_minus1** are further restricted as specified in subclause 8.2.4.3.1.

long_term_pic_num specifies the long-term picture number of the picture being moved to the current index in the list. When decoding a coded frame, **long_term_pic_num** shall be equal to a **LongTermPicNum** assigned to one of the reference frames or complementary reference field pairs marked as "used for long-term reference". When decoding a coded field, **long_term_pic_num** shall be equal to a **LongTermPicNum** assigned to one of the reference fields marked as "used for long-term reference".

7.4.3.2 Prediction weight table semantics

luma_log2_weight_denom is the base 2 logarithm of the denominator for all luma weighting factors. The value of **luma_log2_weight_denom** shall be in the range of 0 to 7, inclusive.

chroma_log2_weight_denom is the base 2 logarithm of the denominator for all chroma weighting factors. The value of **chroma_log2_weight_denom** shall be in the range of 0 to 7, inclusive.

luma_weight_l0_flag equal to 1 specifies that weighting factors for the luma component of list 0 prediction are present. **luma_weight_l0_flag** equal to 0 specifies that these weighting factors are not present.

luma_weight_l0[i] is the weighting factor applied to the luma prediction value for list 0 prediction using **RefPicList0[i]**. When **luma_weight_l0_flag** is equal to 1, the value of **luma_weight_l0[i]** shall be in the range of –128 to 127, inclusive. When **luma_weight_l0_flag** is equal to 0, **luma_weight_l0[i]** shall be inferred to be equal to $2^{\text{luma_log2_weight_denom}}$ for **RefPicList0[i]**.

luma_offset_l0[i] is the additive offset applied to the luma prediction value for list 0 prediction using **RefPicList0[i]**. The value of **luma_offset_l0[i]** shall be in the range of –128 to 127, inclusive. When **luma_weight_l0_flag** is equal to 0, **luma_offset_l0[i]** shall be inferred as equal to 0 for **RefPicList0[i]**.

chroma_weight_l0_flag equal to 1 specifies that weighting factors for the chroma prediction values of list 0 prediction are present. **chroma_weight_l0_flag** equal to 0 specifies that these weighting factors are not present.

chroma_weight_l0[i][j] is the weighting factor applied to the chroma prediction values for list 0 prediction using **RefPicList0[i]** with **j** equal to 0 for Cb and **j** equal to 1 for Cr. When **chroma_weight_l0_flag** is equal to 1, the value of **chroma_weight_l0[i][j]** shall be in the range of –128 to 127, inclusive. When **chroma_weight_l0_flag** is equal to 0, **chroma_weight_l0[i][j]** shall be inferred to be equal to $2^{\text{chroma_log2_weight_denom}}$ for **RefPicList0[i]**.

chroma_offset_l0[i][j] is the additive offset applied to the chroma prediction values for list 0 prediction using **RefPicList0[i]** with **j** equal to 0 for Cb and **j** equal to 1 for Cr. The value of **chroma_offset_l0[i][j]** shall be in the range of –128 to 127, inclusive. When **chroma_weight_l0_flag** is equal to 0, **chroma_offset_l0[i][j]** shall be inferred to be equal to 0 for **RefPicList0[i]**.

luma_weight_l1_flag, **luma_weight_l1**, **luma_offset_l1**, **chroma_weight_l1_flag**, **chroma_weight_l1**, **chroma_offset_l1** have the same semantics as **luma_weight_l0_flag**, **luma_weight_l0**, **luma_offset_l0**, **chroma_weight_l0_flag**, **chroma_weight_l0**, **chroma_offset_l0**, respectively, with **l0**, **list 0**, and **List0** replaced by **l1**, **list 1**, and **List1**, respectively.

7.4.3.3 Decoded reference picture marking semantics

The syntax elements **no_output_of_prior_pics_flag**, **long_term_reference_flag**, **adaptive_ref_pic_marking_mode_flag**, **memory_management_control_operation**, **difference_of_pic_nums_minus1**, **long_term_frame_idx**, **long_term_pic_num**, and **max_long_term_frame_idx_plus1** specify marking of the reference pictures.

The marking of a reference picture can be "unused for reference", "used for short-term reference", or "used for long-term reference", but only one among these three. When a reference picture is referred to as being marked as "used for reference", this collectively refers to the picture being marked as "used for short-term reference" or "used for long-term reference" (but not both). A reference picture that is marked as "used for short-term reference" is referred to as a short-term reference picture. A reference picture that is marked as "used for long-term reference" is referred to as a long-term reference picture.

The syntax element **adaptive_ref_pic_marking_mode_flag** and the content of the decoded reference picture marking syntax structure shall be identical for all coded slices of a coded picture.

The syntax category of the decoded reference picture marking syntax structure shall be inferred as follows.

- If the decoded reference picture marking syntax structure is in a slice header, the syntax category of the decoded reference picture marking syntax structure shall be inferred to be equal to 2.

- Otherwise (the decoded reference picture marking syntax structure is in a decoded reference picture marking repetition SEI message as specified in Annex D), the syntax category of the decoded reference picture marking syntax structure shall be inferred to be equal to 5.

no_output_of_prior_pics_flag specifies how the previously-decoded pictures in the decoded picture buffer are treated after decoding of an IDR picture. See Annex C. When the IDR picture is the first IDR picture in the bitstream, the value of **no_output_of_prior_pics_flag** has no effect on the decoding process. When the IDR picture is not the first IDR picture in the bitstream and the value of **PicWidthInMbs**, **FrameHeightInMbs**, or **max_dec_frame_buffering** derived from the active sequence parameter set is different from the value of **PicWidthInMbs**, **FrameHeightInMbs**, or **max_dec_frame_buffering** derived from the sequence parameter set active for the preceding picture, **no_output_of_prior_pics_flag** equal to 1 may be inferred by the decoder, regardless of the actual value of **no_output_of_prior_pics_flag**.

long_term_reference_flag equal to 0 specifies that the **MaxLongTermFrameIdx** variable is set equal to “no long-term frame indices” and that the IDR picture is marked as “used for short-term reference”. **long_term_reference_flag** equal to 1 specifies that the **MaxLongTermFrameIdx** variable is set equal to 0 and that the current IDR picture is marked “used for long-term reference” and is assigned **LongTermFrameIdx** equal to 0. When **num_ref_frames** is equal to 0, **long_term_reference_flag** shall be equal to 0.

adaptive_ref_pic_marking_mode_flag selects the reference picture marking mode of the currently decoded picture as specified in Table 7-8. **adaptive_ref_pic_marking_mode_flag** shall be equal to 1 when the number of frames, complementary field pairs, and non-paired fields that are currently marked as “used for long-term reference” is equal to $\text{Max}(\text{num_ref_frames}, 1)$.

Table 7-8 – Interpretation of adaptive_ref_pic_marking_mode_flag

adaptive_ref_pic_marking_mode_flag	Reference picture marking mode specified
0	Sliding window reference picture marking mode: A marking mode providing a first-in first-out mechanism for short-term reference pictures.
1	Adaptive reference picture marking mode: A reference picture marking mode providing syntax elements to specify marking of reference pictures as “unused for reference” and to assign long-term frame indices.

memory_management_control_operation specifies a control operation to be applied to affect the reference picture marking. The **memory_management_control_operation** syntax element is followed by data necessary for the operation specified by the value of **memory_management_control_operation**. The values and control operations associated with **memory_management_control_operation** are specified in Table 7-9. The **memory_management_control_operation** syntax elements are processed by the decoding process in the order in which they appear in the slice header, and the semantics constraints expressed for each **memory_management_control_operation** apply at the specific position in that order at which that individual **memory_management_control_operation** is processed.

For interpretation of **memory_management_control_operation**, the term reference picture is interpreted as follows.

- If the current picture is a frame, the term reference picture refers either to a reference frame or a complementary reference field pair.
- Otherwise (the current picture is a field), the term reference picture refers either to a reference field or a field of a reference frame.

memory_management_control_operation shall not be equal to 1 in a slice header unless the specified reference picture is marked as “used for short-term reference” when the **memory_management_control_operation** is processed by the decoding process.

memory_management_control_operation shall not be equal to 2 in a slice header unless the specified long-term picture number refers to a reference picture that is marked as “used for long-term reference” when the **memory_management_control_operation** is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 in a slice header unless the specified reference picture is marked as “used for short-term reference” when the **memory_management_control_operation** is processed by the decoding process.

memory_management_control_operation shall not be equal to 3 or 6 if the value of the variable MaxLongTermFrameIdx is equal to "no long-term frame indices" when the memory_management_control_operation is processed by the decoding process.

Not more than one memory_management_control_operation equal to 4 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 5 shall be present in a slice header.

Not more than one memory_management_control_operation equal to 6 shall be present in a slice header.

memory_management_control_operation shall not be equal to 5 in a slice header unless no memory_management_control_operation in the range of 1 to 3 is present in the same decoded reference picture marking syntax structure.

A memory_management_control_operation equal to 5 shall not follow a memory_management_control_operation equal to 6 in the same slice header.

When a memory_management_control_operation equal to 6 is present, any memory_management_control_operation equal to 2, 3, or 4 that follows the memory_management_control_operation equal to 6 within the same slice header shall not specify the current picture to be marked as "unused for reference".

NOTE 1 – These constraints prohibit any combination of multiple memory_management_control_operation syntax elements that would specify the current picture to be marked as "unused for reference". However, some other combinations of memory_management_control_operation syntax elements are permitted that may affect the marking status of other reference pictures more than once in the same slice header. In particular, it is permitted for a memory_management_control_operation equal to 3 that specifies a long-term frame index to be assigned to a particular short-term reference picture to be followed in the same slice header by a memory_management_control_operation equal to 2, 3, 4 or 6 that specifies the same reference picture to subsequently be marked as "unused for reference".

Table 7-9 – Memory management control operation (memory_management_control_operation) values

memory_management_control_operation	Memory Management Control Operation
0	End memory_management_control_operation syntax element loop
1	Mark a short-term reference picture as "unused for reference"
2	Mark a long-term reference picture as "unused for reference"
3	Mark a short-term reference picture as "used for long-term reference" and assign a long-term frame index to it
4	Specify the maximum long-term frame index and mark all long-term reference pictures having long-term frame indices greater than the maximum value as "unused for reference"
5	Mark all reference pictures as "unused for reference" and set the MaxLongTermFrameIdx variable to "no long-term frame indices"
6	Mark the current picture as "used for long-term reference" and assign a long-term frame index to it

When decoding a field and a memory_management_control_operation command equal to 3 is present that assigns a long-term frame index to a field that is part of a short-term reference frame or part of a short-term complementary reference field pair, another memory_management_control_operation command to assign the same long-term frame index to the other field of the same frame or complementary reference field pair shall be present in the same decoded reference picture marking syntax structure.

NOTE 2 – The above requirement must be fulfilled even when the field referred to by the memory_management_control_operation equal to 3 is subsequently marked as "unused for reference" (for example when a memory_management_control_operation equal to 2 is present in the same slice header that causes the field to be marked as "unused for reference").

When the first field (in decoding order) of a complementary reference field pair includes a `long_term_reference_flag` equal to 1 or a `memory_management_control_operation` command equal to 6, the decoded reference picture marking syntax structure for the other field of the complementary reference field pair shall contain a `memory_management_control_operation` command equal to 6 that assigns the same long-term frame index to the other field.

NOTE 3 – The above requirement must be fulfilled even when the first field of the complementary reference field pair is subsequently marked as "unused for reference" (for example, when a `memory_management_control_operation` equal to 2 is present in the slice header of the second field that causes the first field to be marked as "unused for reference").

`difference_of_pic_nums_minus1` is used (with `memory_management_control_operation` equal to 3 or 1) to assign a long-term frame index to a short-term reference picture or to mark a short-term reference picture as "unused for reference". When the associated `memory_management_control_operation` is processed by the decoding process, the resulting picture number derived from `difference_of_pic_nums_minus1` shall be a picture number assigned to one of the reference pictures marked as "used for reference" and not previously assigned to a long-term frame index.

The resulting picture number is constrained as follows.

- If `field_pic_flag` is equal to 0, the resulting picture number shall be one of the set of picture numbers assigned to reference frames or complementary reference field pairs.

NOTE 4 – When `field_pic_flag` is equal to 0, the resulting picture number must be a picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when `field_pic_flag` is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a `memory_management_control_operation` equal to 1.

- Otherwise (`field_pic_flag` is equal to 1), the resulting picture number shall be one of the set of picture numbers assigned to reference fields.

`long_term_pic_num` is used (with `memory_management_control_operation` equal to 2) to mark a long-term reference picture as "unused for reference". When the associated `memory_management_control_operation` is processed by the decoding process, `long_term_pic_num` shall be equal to a long-term picture number assigned to one of the reference pictures that is currently marked as "used for long-term reference".

The resulting long-term picture number is constrained as follows.

- If `field_pic_flag` is equal to 0, the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference frames or complementary reference field pairs.

NOTE 5 – When `field_pic_flag` is equal to 0, the resulting long-term picture number must be a long-term picture number assigned to a complementary reference field pair in which both fields are marked as "used for reference" or a frame in which both fields are marked as "used for reference". In particular, when `field_pic_flag` is equal to 0, the marking of a non-paired field or a frame in which a single field is marked as "used for reference" cannot be affected by a `memory_management_control_operation` equal to 2.

- Otherwise (`field_pic_flag` is equal to 1), the resulting long-term picture number shall be one of the set of long-term picture numbers assigned to reference fields.

`long_term_frame_idx` is used (with `memory_management_control_operation` equal to 3 or 6) to assign a long-term frame index to a picture. When the associated `memory_management_control_operation` is processed by the decoding process, the value of `long_term_frame_idx` shall be in the range of 0 to `MaxLongTermFrameIdx`, inclusive.

`max_long_term_frame_idx_plus1` minus 1 specifies the maximum value of long-term frame index allowed for long-term reference pictures (until receipt of another value of `max_long_term_frame_idx_plus1`). The value of `max_long_term_frame_idx_plus1` shall be in the range of 0 to `num_ref_frames`, inclusive.

7.4.4 Slice data semantics

`cabac_alignment_one_bit` is a bit equal to 1.

`mb_skip_run` specifies the number of consecutive skipped macroblocks for which, when decoding a P or SP slice, `mb_type` shall be inferred to be `P_Skip` and the macroblock type is collectively referred to as a P macroblock type, or for which, when decoding a B slice, `mb_type` shall be inferred to be `B_Skip` and the macroblock type is collectively referred to as a B macroblock type. The value of `mb_skip_run` shall be in the range of 0 to `PicSizeInMbs – CurrMbAddr`, inclusive.

`mb_skip_flag` equal to 1 specifies that for the current macroblock, when decoding a P or SP slice, `mb_type` shall be inferred to be `P_Skip` and the macroblock type is collectively referred to as P macroblock type, or for which, when decoding a B slice, `mb_type` shall be inferred to be `B_Skip` and the macroblock type is collectively referred to as B macroblock type. `mb_skip_flag` equal to 0 specifies that the current macroblock is not skipped.

mb_field_decoding_flag equal to 0 specifies that the current macroblock pair is a frame macroblock pair. **mb_field_decoding_flag** equal to 1 specifies that the macroblock pair is a field macroblock pair. Both macroblocks of a frame macroblock pair are referred to in the text as frame macroblocks, whereas both macroblocks of a field macroblock pair are referred to in the text as field macroblocks.

When **mb_field_decoding_flag** is not present for either macroblock of a macroblock pair, the value of **mb_field_decoding_flag** is derived as follows.

- If there is a neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice, the value of **mb_field_decoding_flag** shall be inferred to be equal to the value of **mb_field_decoding_flag** for the neighbouring macroblock pair immediately to the left of the current macroblock pair.
- Otherwise, if there is no neighbouring macroblock pair immediately to the left of the current macroblock pair in the same slice and there is a neighbouring macroblock pair immediately above the current macroblock pair in the same slice, the value of **mb_field_decoding_flag** shall be inferred to be equal to the value of **mb_field_decoding_flag** for the neighbouring macroblock pair immediately above the current macroblock pair.
- Otherwise (there is no neighbouring macroblock pair either immediately to the left or immediately above the current macroblock pair in the same slice), the value of **mb_field_decoding_flag** shall be inferred to be equal to 0.

end_of_slice_flag equal to 0 specifies that another macroblock is following in the slice. **end_of_slice_flag** equal to 1 specifies the end of the slice and that no further macroblock follows.

The function `NextMbAddress()` used in the slice data syntax table is specified in subclause 8.2.2.

7.4.5 Macroblock layer semantics

mb_type specifies the macroblock type. The semantics of **mb_type** depend on the slice type.

Tables and semantics are specified for the various macroblock types for I, SI, P, SP, and B slices. Each table presents the value of **mb_type**, the name of **mb_type**, the number of macroblock partitions used (given by the `NumMbPart(mb_type)` function), the prediction mode of the macroblock (when it is not partitioned) or the first partition (given by the `MbPartPredMode(mb_type, 0)` function) and the prediction mode of the second partition (given by the `MbPartPredMode(mb_type, 1)` function). When a value is not applicable it is designated by “na”. In the text, the value of **mb_type** may be referred to as the macroblock type and a value X of `MbPartPredMode()` may be referred to in the text by “X macroblock (partition) prediction mode” or as “X prediction macroblocks”.

Table 7-10 shows the allowed collective macroblock types for each slice_type.

NOTE 1 – There are some macroblock types with `Pred_L0` prediction mode that are classified as B macroblock types.

Table 7-10 – Allowed collective macroblock types for slice_type

slice_type	Allowed collective macroblock types
I (slice)	I (see Table 7-11) (macroblock types)
P (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)
B (slice)	B (see Table 7-14) and I (see Table 7-11) (macroblock types)
SI (slice)	SI (see Table 7-12) and I (see Table 7-11) (macroblock types)
SP (slice)	P (see Table 7-13) and I (see Table 7-11) (macroblock types)

transform_size_8x8_flag equal to 1 specifies that for the current macroblock the transform coefficient decoding process and picture construction process prior to deblocking filter process for residual 8x8 blocks shall be invoked for luma samples, and when `ChromaArrayType == 3` also for Cb and Cr samples. **transform_size_8x8_flag** equal to 0 specifies that for the current macroblock the transform coefficient decoding process and picture construction process prior to deblocking filter process for residual 4x4 blocks shall be invoked for luma samples, and when `ChromaArrayType == 3` also for Cb and Cr samples. When **transform_size_8x8_flag** is not present in the bitstream, it shall be inferred to be equal to 0.

NOTE 2 – When the current macroblock prediction mode `MbPartPredMode(mb_type, 0)` is equal to `Intra_16x16`, **transform_size_8x8_flag** is not present in the bitstream and then inferred to be equal to 0.

When `sub_mb_type[mbPartIdx]` (see subclause 7.4.5.2) is present in the bitstream for all 8x8 blocks indexed by `mbPartIdx = 0..3`, the variable `noSubMbPartSizeLessThan8x8Flag` indicates whether for each of the four 8x8 blocks the corresponding `SubMbPartWidth(sub_mb_type[mbPartIdx])` and `SubMbPartHeight(sub_mb_type[mbPartIdx])` are both equal to 8.

NOTE 3 – When noSubMbPartSizeLessThan8x8Flag is equal to 0 and the current macroblock type is not equal to I_NxN, transform_size_8x8_flag is not present in the bitstream and then inferred to be equal to 0.

Macroblock types that may be collectively referred to as I macroblock types are specified in Table 7-11.

The macroblock types for I slices are all I macroblock types.

Table 7-11 – Macroblock types for I slices

mb_type	Name of mb_type	transform_size_8x8_flag	MbPartPredMode (mb_type, 0)	Intra16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	I_NxN	0	Intra_4x4	na	Equation 7-34	Equation 7-34
0	I_NxN	1	Intra_8x8	na	Equation 7-34	Equation 7-34
1	I_16x16_0_0_0	na	Intra_16x16	0	0	0
2	I_16x16_1_0_0	na	Intra_16x16	1	0	0
3	I_16x16_2_0_0	na	Intra_16x16	2	0	0
4	I_16x16_3_0_0	na	Intra_16x16	3	0	0
5	I_16x16_0_1_0	na	Intra_16x16	0	1	0
6	I_16x16_1_1_0	na	Intra_16x16	1	1	0
7	I_16x16_2_1_0	na	Intra_16x16	2	1	0
8	I_16x16_3_1_0	na	Intra_16x16	3	1	0
9	I_16x16_0_2_0	na	Intra_16x16	0	2	0
10	I_16x16_1_2_0	na	Intra_16x16	1	2	0
11	I_16x16_2_2_0	na	Intra_16x16	2	2	0
12	I_16x16_3_2_0	na	Intra_16x16	3	2	0
13	I_16x16_0_0_1	na	Intra_16x16	0	0	15
14	I_16x16_1_0_1	na	Intra_16x16	1	0	15
15	I_16x16_2_0_1	na	Intra_16x16	2	0	15
16	I_16x16_3_0_1	na	Intra_16x16	3	0	15
17	I_16x16_0_1_1	na	Intra_16x16	0	1	15
18	I_16x16_1_1_1	na	Intra_16x16	1	1	15
19	I_16x16_2_1_1	na	Intra_16x16	2	1	15
20	I_16x16_3_1_1	na	Intra_16x16	3	1	15
21	I_16x16_0_2_1	na	Intra_16x16	0	2	15
22	I_16x16_1_2_1	na	Intra_16x16	1	2	15
23	I_16x16_2_2_1	na	Intra_16x16	2	2	15
24	I_16x16_3_2_1	na	Intra_16x16	3	2	15
25	I_PCM	na	na	na	na	na

The following semantics are assigned to the macroblock types in Table 7-11.

I_NxN: A mnemonic name for mb_type equal to 0 with MbPartPredMode(mb_type, 0) equal to Intra_4x4 or Intra_8x8.

I_16x16_0_0_0, I_16x16_1_0_0, I_16x16_2_0_0, I_16x16_3_0_0, I_16x16_0_1_0, I_16x16_1_1_0, I_16x16_2_1_0, I_16x16_3_1_0, I_16x16_0_2_0, I_16x16_1_2_0, I_16x16_2_2_0, I_16x16_3_2_0, I_16x16_0_0_1, I_16x16_1_0_1, I_16x16_2_0_1, I_16x16_3_0_1, I_16x16_0_1_1, I_16x16_1_1_1, I_16x16_2_1_1, I_16x16_3_1_1, I_16x16_0_2_1, I_16x16_1_2_1, I_16x16_2_2_1, I_16x16_3_2_1: the macroblock is coded as an Intra_16x16 prediction mode macroblock.

To each Intra 16x16 prediction macroblock, an Intra16x16PredMode is assigned, which specifies the Intra 16x16 prediction mode, and values of CodedBlockPatternLuma and CodedBlockPatternChroma are assigned as specified in Table 7-11.

Intra_4x4 specifies the macroblock prediction mode and specifies that the Intra_4x4 prediction process is invoked as specified in subclause 8.3.1. Intra_4x4 is an Intra macroblock prediction mode.

Intra_8x8 specifies the macroblock prediction mode and specifies that the Intra_8x8 prediction process is invoked as specified in subclause 8.3.2. Intra_8x8 is an Intra macroblock prediction mode.

Intra_16x16 specifies the macroblock prediction mode and specifies that the Intra_16x16 prediction process is invoked as specified in subclause 8.3.3. Intra_16x16 is an Intra macroblock prediction mode.

For a macroblock coded with mb_type equal to I_PCM, the Intra macroblock prediction mode shall be inferred.

A macroblock type that may be referred to as SI macroblock type is specified in Table 7-12.

The macroblock types for SI slices are specified in Table 7-12 and 7-11. The mb_type value 0 is specified in Table 7-12 and the mb_type values 1 to 26 are specified in Table 7-11, indexed by subtracting 1 from the value of mb_type.

Table 7-12 – Macroblock type with value 0 for SI slices

mb_type	Name of mb_type	MbPartPredMode (mb_type, 0)	Intra 16x16PredMode	CodedBlockPatternChroma	CodedBlockPatternLuma
0	SI	Intra_4x4	na	Equation 7-34	Equation 7-34

The following semantics are assigned to the macroblock type in Table 7-12. The SI macroblock is coded as Intra_4x4 prediction macroblock.

Macroblock types that may be collectively referred to as P macroblock types are specified in Table 7-13.

The macroblock types for P and SP slices are specified in Table 7-13 and 7-11. mb_type values 0 to 4 are specified in Table 7-13 and mb_type values 5 to 30 are specified in Table 7-11, indexed by subtracting 5 from the value of mb_type.

Table 7-13 – Macroblock type values 0 to 4 for P and SP slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	P_L0_16x16	1	Pred_L0	na	16	16
1	P_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
2	P_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
3	P_8x8	4	na	na	8	8
4	P_8x8ref0	4	na	na	8	8
inferred	P_Skip	1	Pred_L0	na	16	16

The following semantics are assigned to the macroblock types in Table 7-13.

- P_L0_16x16: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples.
- P_L0_L0_MxN, with MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively.
- P_8x8: for each sub-macroblock an additional syntax element (sub_mb_type) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).
- P_8x8ref0: has the same semantics as P_8x8 but no syntax element for the reference index (ref_idx_l0) is present in the bitstream and ref_idx_l0[mbPartIdx] shall be inferred to be equal to 0 for all sub-macroblocks of the macroblock (with indices mbPartIdx equal to 0..3).
- P_Skip: no further data is present for the macroblock in the bitstream.

The following semantics are assigned to the macroblock prediction modes (MbPartPredMode()) in Table 7-13.

- Pred_L0: specifies that the inter prediction process is invoked using list 0 prediction. Pred_L0 is an Inter macroblock prediction mode.

Macroblock types that may be collectively referred to as B macroblock types are specified in Table 7-14.

The macroblock types for B slices are specified in Table 7-14 and 7-11. The mb_type values 0 to 22 are specified in Table 7-14 and the mb_type values 23 to 48 are specified in Table 7-11, indexed by subtracting 23 from the value of mb_type.

Table 7-14 – Macroblock type values 0 to 22 for B slices

mb_type	Name of mb_type	NumMbPart (mb_type)	MbPartPredMode (mb_type, 0)	MbPartPredMode (mb_type, 1)	MbPartWidth (mb_type)	MbPartHeight (mb_type)
0	B_Direct_16x16	na	Direct	na	8	8
1	B_L0_16x16	1	Pred_L0	na	16	16
2	B_L1_16x16	1	Pred_L1	na	16	16
3	B_Bi_16x16	1	BiPred	na	16	16
4	B_L0_L0_16x8	2	Pred_L0	Pred_L0	16	8
5	B_L0_L0_8x16	2	Pred_L0	Pred_L0	8	16
6	B_L1_L1_16x8	2	Pred_L1	Pred_L1	16	8
7	B_L1_L1_8x16	2	Pred_L1	Pred_L1	8	16
8	B_L0_L1_16x8	2	Pred_L0	Pred_L1	16	8
9	B_L0_L1_8x16	2	Pred_L0	Pred_L1	8	16
10	B_L1_L0_16x8	2	Pred_L1	Pred_L0	16	8
11	B_L1_L0_8x16	2	Pred_L1	Pred_L0	8	16
12	B_L0_Bi_16x8	2	Pred_L0	BiPred	16	8
13	B_L0_Bi_8x16	2	Pred_L0	BiPred	8	16
14	B_L1_Bi_16x8	2	Pred_L1	BiPred	16	8
15	B_L1_Bi_8x16	2	Pred_L1	BiPred	8	16
16	B_Bi_L0_16x8	2	BiPred	Pred_L0	16	8
17	B_Bi_L0_8x16	2	BiPred	Pred_L0	8	16
18	B_Bi_L1_16x8	2	BiPred	Pred_L1	16	8
19	B_Bi_L1_8x16	2	BiPred	Pred_L1	8	16
20	B_Bi_Bi_16x8	2	BiPred	BiPred	16	8
21	B_Bi_Bi_8x16	2	BiPred	BiPred	8	16
22	B_8x8	4	na	na	8	8
inferred	B_Skip	na	Direct	na	8	8

The following semantics are assigned to the macroblock types in Table 7-14:

- B_Direct_16x16: no motion vector differences or reference indices are present for the macroblock in the bitstream. The functions MbPartWidth(B_Direct_16x16), and MbPartHeight(B_Direct_16x16) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

- B_X_16x16 with X being replaced by L0, L1, or Bi: the samples of the macroblock are predicted with one luma macroblock partition of size 16x16 luma samples and associated chroma samples. For a macroblock with type B_X_16x16 with X being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream for the macroblock. For a macroblock with type B_X_16x16 with X being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock.
- B_X0_X1_MxN, with X0, X1 referring to the first and second macroblock partition and being replaced by L0, L1, or Bi, and MxN being replaced by 16x8 or 8x16: the samples of the macroblock are predicted using two luma partitions of size MxN equal to 16x8, or two luma partitions of size MxN equal to 8x16, and associated chroma samples, respectively. For a macroblock partition X0 or X1 with X0 or X1 being replaced by either L0 or L1, one motion vector difference and one reference index is present in the bitstream. For a macroblock partition X0 or X1 with X0 or X1 being replaced by Bi, two motion vector differences and two reference indices are present in the bitstream for the macroblock partition.
- B_8x8: for each sub-macroblock an additional syntax element (sub_mb_type) is present in the bitstream that specifies the type of the corresponding sub-macroblock (see subclause 7.4.5.2).
- B_Skip: no further data is present for the macroblock in the bitstream. The functions MbPartWidth(B_Skip), and MbPartHeight(B_Skip) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.

The following semantics are assigned to the macroblock prediction modes (MbPartPredMode()) in Table 7-14.

- Direct: no motion vector differences or reference indices are present for the macroblock (in case of B_Skip or B_Direct_16x16) in the bitstream. Direct is an Inter macroblock prediction mode.
- Pred_L0: see semantics for Table 7-13.
- Pred_L1: specifies that the Inter prediction process is invoked using list 1 prediction. Pred_L1 is an Inter macroblock prediction mode.
- BiPred: specifies that the Inter prediction process is invoked using list 0 and list 1 prediction. BiPred is an Inter macroblock prediction mode.

pcm_alignment_zero_bit is a bit equal to 0.

pcm_sample_luma[i] is a sample value. The first pcm_sample_luma[i] values represent luma sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is BitDepth_Y. When profile_idc is not equal to 44, 100, 110, 122, or 244, pcm_sample_luma[i] shall not be equal to 0.

pcm_sample_chroma[i] is a sample value. The first MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cb sample values in the raster scan within the macroblock and the remaining MbWidthC * MbHeightC pcm_sample_chroma[i] values represent Cr sample values in the raster scan within the macroblock. The number of bits used to represent each of these samples is BitDepth_C. When profile_idc is not equal to 44, 100, 110, 122, or 244, pcm_sample_chroma[i] shall not be equal to 0.

coded_block_pattern specifies which of the four 8x8 luma blocks and associated chroma blocks of a macroblock may contain non-zero transform coefficient levels. When coded_block_pattern is present in the bitstream, the variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived as follows.

$$\begin{aligned} \text{CodedBlockPatternLuma} &= \text{coded_block_pattern} \% 16 \\ \text{CodedBlockPatternChroma} &= \text{coded_block_pattern} / 16 \end{aligned} \quad (7-34)$$

When the macroblock type is not equal to I_PCM, the following applies.

- If the macroblock prediction mode is equal Intra_16x16, the following applies.
 - If ChromaArrayType is not equal to 3, the value of CodedBlockPatternLuma distinguishes between the following cases.
 - If CodedBlockPatternLuma is equal to 0, all AC transform coefficient levels of the luma component of the macroblock are equal to 0 for all 16 of the 4x4 blocks in the 16x16 luma block.
 - Otherwise (CodedBlockPatternLuma is not equal to 0), CodedBlockPatternLuma is equal to 15, at least one of the AC transform coefficient levels of the luma component of the macroblock shall be non-zero, and the AC transform coefficient levels are scanned for all 16 of the 4x4 blocks in the 16x16 block.

- Otherwise (ChromaArrayType is equal to 3), the value of CodedBlockPatternLuma distinguishes between the following cases.
 - If CodedBlockPatternLuma is equal to 0, all AC transform coefficient levels of the luma, Cb, and Cr components of the macroblock are equal to 0 for all 16 of the 4x4 blocks in the luma, Cb, and Cr components of the macroblock.
 - Otherwise (CodedBlockPatternLuma is not equal to 0), CodedBlockPatternLuma is equal to 15, at least one of the AC transform coefficient levels of the luma, Cb, or Cr components of the macroblock shall be non-zero, and the AC transform coefficient levels are scanned for all 16 of the 4x4 blocks in the luma, Cb, and Cr components of the macroblock.
- Otherwise (the macroblock prediction mode is not equal to Intra_16x16), coded_block_pattern is present in the bitstream, and the following applies.
 - If ChromaArrayType is not equal to 3, each of the four LSBs of CodedBlockPatternLuma distinguishes, for one of the four 8x8 luma blocks of the macroblock, between the following cases.
 - If the corresponding bit of CodedBlockPatternLuma is equal to 0, all transform coefficient levels of the luma transform blocks in the 8x8 luma block are equal to zero.
 - Otherwise (the corresponding bit of CodedBlockPatternLuma is equal to 1), one or more transform coefficient levels of one or more of the luma transform blocks in the 8x8 luma block shall be non-zero valued and the transform coefficient levels of the corresponding transform blocks are scanned.
 - Otherwise (ChromaArrayType is equal to 3), each of the four LSBs of CodedBlockPatternLuma distinguishes, for one of the four 8x8 luma blocks of the macroblock, between the following cases.
 - If the corresponding bit of CodedBlockPatternLuma is equal to 0, all transform coefficient levels of the luma, Cb, and Cr transform blocks in the 8x8 luma block are equal to zero.
 - Otherwise (the corresponding bit of CodedBlockPatternLuma is equal to 1), one or more transform coefficient levels of one or more of the luma, Cb, or Cr transform blocks in the 8x8 luma block shall be non-zero valued and the transform coefficient levels of the corresponding transform blocks are scanned.

When the macroblock type is not equal to I_PCM, CodedBlockPatternChroma is interpreted as follows.

If ChromaArrayType is not equal to 0 or 3, CodedBlockPatternChroma is specified in Table 7-15.

- Otherwise (ChromaArrayType is equal to 0 or 3), the bitstream shall not contain data that results in a derived value of CodedBlockPatternChroma that is not equal to 0.

Table 7-15 – Specification of CodedBlockPatternChroma values

CodedBlockPatternChroma	Description
0	All chroma transform coefficient levels are equal to 0.
1	One or more chroma DC transform coefficient levels shall be non-zero valued. All chroma AC transform coefficient levels are equal to 0.
2	Zero or more chroma DC transform coefficient levels are non-zero valued. One or more chroma AC transform coefficient levels shall be non-zero valued.

mb_qp_delta can change the value of QP_Y in the macroblock layer. The decoded value of **mb_qp_delta** shall be in the range of $-(26 + QpBdOffset_Y / 2)$ to $+(25 + QpBdOffset_Y / 2)$, inclusive. **mb_qp_delta** shall be inferred to be equal to 0 when it is not present for any macroblock (including P_Skip and B_Skip macroblock types).

The value of QP_Y is derived as

$$QP_Y = ((QP_{Y,PREV} + mb_qp_delta + 52 + 2 * QpBdOffset_Y) \% (52 + QpBdOffset_Y)) - QpBdOffset_Y \quad (7-35)$$

where $QP_{Y,PREV}$ is the luma quantisation parameter, QP_Y , of the previous macroblock in decoding order in the current slice. For the first macroblock in the slice $QP_{Y,PREV}$ is initially set equal to $SliceQP_Y$ derived in Equation 7-28 at the start of each slice.

The value of QP'_Y is derived as

$$QP'_Y = QP_Y + QpBdOffset_Y \quad (7-36)$$

7.4.5.1 Macroblock prediction semantics

All samples of the macroblock are predicted. The prediction modes are derived using the following syntax elements.

prev_intra4x4_pred_mode_flag[luma4x4BlkIdx] and **rem_intra4x4_pred_mode**[luma4x4BlkIdx] specify the Intra_4x4 prediction of the 4x4 luma block with index luma4x4BlkIdx = 0..15. When ChromaArrayType is equal to 3, **prev_intra4x4_pred_mode_flag**[luma4x4BlkIdx] and **rem_intra4x4_pred_mode**[luma4x4BlkIdx] also specify the Intra_4x4 prediction of the 4x4 Cb block with luma4x4BlkIdx equal to cb4x4BlkIdx for cb4x4BlkIdx = 0..15 and the Intra_4x4 prediction of the 4x4 Cr block with luma4x4BlkIdx equal to cr4x4BlkIdx for cr4x4BlkIdx = 0..15.

prev_intra8x8_pred_mode_flag[luma8x8BlkIdx] and **rem_intra8x8_pred_mode**[luma8x8BlkIdx] specify the Intra_8x8 prediction of the 8x8 luma block with index luma8x8BlkIdx = 0..3. When ChromaArrayType is equal to 3, **prev_intra8x8_pred_mode_flag**[luma8x8BlkIdx] and **rem_intra8x8_pred_mode**[luma8x8BlkIdx] also specify the Intra_8x8 prediction of the 8x8 Cb block with luma8x8BlkIdx equal to cb8x8BlkIdx for cb8x8BlkIdx = 0..3 and the Intra_8x8 prediction of the 8x8 Cr block with index luma8x8BlkIdx equal to cr8x8BlkIdx for cr8x8BlkIdx = 0..3.

intra_chroma_pred_mode specifies the type of spatial prediction used for chroma in macroblocks using Intra_4x4 or Intra_16x16 prediction, as shown in Table 7-16. The value of intra_chroma_pred_mode shall be in the range of 0 to 3, inclusive.

Table 7-16 – Relationship between intra_chroma_pred_mode and spatial prediction modes

intra_chroma_pred_mode	Intra Chroma Prediction Mode
0	DC
1	Horizontal
2	Vertical
3	Plane

ref_idx_l0[mbPartIdx] when present, specifies the index in reference picture list 0 of the reference picture to be used for prediction.

The range of **ref_idx_l0**[mbPartIdx], the index in list 0 of the reference picture, and, if applicable, the parity of the field within the reference picture used for prediction are specified as follows.

- If MbaffFrameFlag is equal to 0 or mb_field_decoding_flag is equal to 0, the value of **ref_idx_l0**[mbPartIdx] shall be in the range of 0 to num_ref_idx_l0_active_minus1, inclusive.
- Otherwise (MbaffFrameFlag is equal to 1 and mb_field_decoding_flag is equal to 1), the value of **ref_idx_l0**[mbPartIdx] shall be in the range of 0 to 2 * num_ref_idx_l0_active_minus1 + 1, inclusive.

When only one reference picture is used for inter prediction, the values of **ref_idx_l0**[mbPartIdx] shall be inferred to be equal to 0.

ref_idx_l1[mbPartIdx] has the same semantics as **ref_idx_l0**, with l0 and list 0 replaced by l1 and list 1, respectively.

mvd_l0[mbPartIdx][0][compIdx] specifies the difference between a vector component to be used and its prediction. The index mbPartIdx specifies to which macroblock partition mvd_l0 is assigned. The partitioning of the macroblock is specified by mb_type. The horizontal motion vector component difference is decoded first in decoding order and is assigned CompIdx = 0. The vertical motion vector component is decoded second in decoding order and is assigned CompIdx = 1. The range of the components of **mvd_l0**[mbPartIdx][0][compIdx] is specified by constraints on the motion vector variable values derived from it as specified in Annex A.

mvd_l1[mbPartIdx][0][compIdx] has the same semantics as **mvd_l0**, with l0 and L0 replaced by l1 and L1, respectively.

7.4.5.2 Sub-macroblock prediction semantics

sub_mb_type[mbPartIdx] specifies the sub-macroblock types.

Tables and semantics are specified for the various sub-macroblock types for P, and B macroblock types. Each table presents the value of sub_mb_type, the name of sub_mb_type, the number of sub-macroblock partitions used (given by the NumSubMbPart(sub_mb_type) function), and the prediction mode of the sub-macroblock (given by the SubMbPredMode(sub_mb_type) function). In the text, the value of sub_mb_type may be referred to by

“sub-macroblock type”. In the text, the value of SubMbPredMode() may be referred to by “sub-macroblock prediction mode”.

The interpretation of sub_mb_type[mbPartIdx] for P macroblock types is specified in Table 7-17, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present.

Table 7-17 – Sub-macroblock types in P macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	na	na	na	na	na
0	P_L0_8x8	1	Pred_L0	8	8
1	P_L0_8x4	2	Pred_L0	8	4
2	P_L0_4x8	2	Pred_L0	4	8
3	P_L0_4x4	4	Pred_L0	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-17.

- P_L0_MxN, with MxN being replaced by 8x8, 8x4, 4x8, or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, two luma partitions of size MxN equal to 8x4, or two luma partitions of size MxN equal to 4x8, or four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively.

The following semantics are assigned to the sub-macroblock prediction modes (SubMbPredMode()) in Table 7-17.

- Pred_L0: see semantics for Table 7-13.

The interpretation of sub_mb_type[mbPartIdx] for B macroblock types is specified in Table 7-18, where the row for "inferred" specifies values inferred when sub_mb_type[mbPartIdx] is not present, and the inferred value "mb_type" specifies that the name of sub_mb_type[mbPartIdx] is the same as the name of mb_type for this case.

Table 7-18 – Sub-macroblock types in B macroblocks

sub_mb_type[mbPartIdx]	Name of sub_mb_type[mbPartIdx]	NumSubMbPart (sub_mb_type[mbPartIdx])	SubMbPredMode (sub_mb_type[mbPartIdx])	SubMbPartWidth (sub_mb_type[mbPartIdx])	SubMbPartHeight (sub_mb_type[mbPartIdx])
inferred	mb_type	4	Direct	4	4
0	B_Direct_8x8	4	Direct	4	4
1	B_L0_8x8	1	Pred_L0	8	8
2	B_L1_8x8	1	Pred_L1	8	8
3	B_Bi_8x8	1	BiPred	8	8
4	B_L0_8x4	2	Pred_L0	8	4
5	B_L0_4x8	2	Pred_L0	4	8
6	B_L1_8x4	2	Pred_L1	8	4
7	B_L1_4x8	2	Pred_L1	4	8
8	B_Bi_8x4	2	BiPred	8	4
9	B_Bi_4x8	2	BiPred	4	8
10	B_L0_4x4	4	Pred_L0	4	4
11	B_L1_4x4	4	Pred_L1	4	4
12	B_Bi_4x4	4	BiPred	4	4

The following semantics are assigned to the sub-macroblock types in Table 7-18:

- B_Skip and B_Direct_16x16: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth() and SubMbPartHeight() are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.
- B_Direct_8x8: no motion vector differences or reference indices are present for the sub-macroblock in the bitstream. The functions SubMbPartWidth(B_Direct_8x8) and SubMbPartHeight(B_Direct_8x8) are used in the derivation process for motion vectors and reference frame indices in subclause 8.4.1 for direct mode prediction.
- B_X_MxN, with X being replaced by L0, L1, or Bi, and MxN being replaced by 8x8, 8x4, 4x8 or 4x4: the samples of the sub-macroblock are predicted using one luma partition of size MxN equal to 8x8, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 8x4, or the samples of the sub-macroblock are predicted using two luma partitions of size MxN equal to 4x8, or the samples of the sub-macroblock are predicted using four luma partitions of size MxN equal to 4x4, and associated chroma samples, respectively. All sub-macroblock partitions share the same reference index. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type being B_X_MxN with X being replaced by either L0 or L1, one motion vector difference is present in the bitstream. For an MxN sub-macroblock partition in a sub-macroblock with sub_mb_type being B_Bi_MxN, two motion vector difference are present in the bitstream.

The following semantics are assigned to the sub-macroblock prediction modes (SubMbPredMode()) in Table 7-18.

- Direct: see semantics for Table 7-14.
- Pred_L0: see semantics for Table 7-13.

- `Pred_L1`: see semantics for Table 7-14.
- `BiPred`: see semantics for Table 7-14.

`ref_idx_l0[ mbPartIdx ]` has the same semantics as `ref_idx_l0` in subclause 7.4.5.1.

`ref_idx_l1[ mbPartIdx ]` has the same semantics as `ref_idx_l1` in subclause 7.4.5.1.

`mvd_l0[ mbPartIdx ][ subMbPartIdx ][ compIdx ]` has the same semantics as `mvd_l0` in subclause 7.4.5.1, except that it is applied to the sub-macroblock partition index with `subMbPartIdx`. The indices `mbPartIdx` and `subMbPartIdx` specify to which macroblock partition and sub-macroblock partition `mvd_l0` is assigned.

`mvd_l1[ mbPartIdx ][ subMbPartIdx ][ compIdx ]` has the same semantics as `mvd_l1` in subclause 7.4.5.1.

7.4.5.3 Residual data semantics

The syntax structure `residual_block()`, which is used for parsing the transform coefficient levels, is assigned as follows.

- If `entropy_coding_mode_flag` is equal to 0, `residual_block` is set equal to `residual_block_cavlc`, which is used for parsing the syntax elements for transform coefficient levels.
- Otherwise (`entropy_coding_mode_flag` is equal to 1), `residual_block` is set equal to `residual_block_cabac`, which is used for parsing the syntax elements for transform coefficient levels.

The syntax structure `residual_luma( i16x16DClevel, i16x16AClevel, level, level8x8 )` is used with the variables in brackets being its output and being assigned as follows.

`Intra16x16DClevel` is set equal to `i16x16DClevel`, `Intra16x16AClevel` is set equal to `i16x16AClevel`, `LumaLevel` is set equal to `level`, and `LumaLevel8x8` is set equal to `level8x8`.

When `ChromaArrayType` is equal to 1 or 2, the following applies.

- For each chroma component, indexed by `iCbCr = 0..1`, the DC transform coefficient levels of the $4 * \text{NumC8x8}$ 4x4 chroma blocks are parsed into the `iCbCr`-th list `ChromaDClevel[ iCbCr ]`.
- For each of the 4x4 chroma blocks, indexed by `i4x4 = 0..3` and `i8x8 = 0..NumC8x8 - 1`, of each chroma component, indexed by `iCbCr = 0..1`, the 15 AC transform coefficient levels are parsed into the $(i8x8*4 + i4x4)$ -th list of the `iCbCr`-th chroma component `ChromaAClevel[ iCbCr ][ i8x8*4 + i4x4 ]`.

When `ChromaArrayType` is equal to 3, the following applies.

- The syntax structure `residual_luma( i16x16DClevel, i16x16AClevel, level, level8x8 )` is used for the Cb component with the variables in brackets being its output and being assigned as follows. `CbIntra16x16DClevel` is set equal to `i16x16DClevel`, `CbIntra16x16AClevel` is set equal to `i16x16AClevel`, `CbLevel` is set equal to `level`, and `CbLevel8x8` is set equal to `level8x8`.
- The syntax structure `residual_luma( i16x16DClevel, i16x16AClevel, level, level8x8 )` is used for the Cr component with the variables in brackets being its output and being assigned as follows. `CrIntra16x16DClevel` is set equal to `i16x16DClevel`, `CrIntra16x16AClevel` is set equal to `i16x16AClevel`, `CrLevel` is set equal to `level`, and `CrLevel8x8` is set equal to `level8x8`.

7.4.5.3.1 Residual luma data semantics

Output of this syntax structure are the variables `i16x16DClevel`, `i16x16AClevel`, `level`, and `level8x8`.

The syntax structure `residual_block()`, which is used for parsing the transform coefficient levels, is assigned as follows.

- If `entropy_coding_mode_flag` is equal to 0, `residual_block` is set equal to `residual_block_cavlc`, which is used for parsing the syntax elements for transform coefficient levels.
- Otherwise (`entropy_coding_mode_flag` is equal to 1), `residual_block` is set equal to `residual_block_cabac`, which is used for parsing the syntax elements for transform coefficient levels.

Depending on `mb_type`, the syntax structure `residual_block( coeffLevel, maxNumCoeff )` is used with the arguments `coeffLevel`, which is a list containing the `maxNumCoeff` transform coefficient levels that are parsed in `residual_block()` and `maxNumCoeff` as follows.

Depending on `MbPartPredMode( mb_type, 0 )`, the following applies.

- If `MbPartPredMode( mb_type, 0 )` is equal to `Intra_16x16`, the transform coefficient levels are parsed into the list `i16x16DClevel` and into the 16 lists `i16x16AClevel[ i ]`. `i16x16DClevel` contains the 16 transform coefficient levels

of the DC transform coefficient levels for each 4x4 luma block. For each of the 16 4x4 luma blocks indexed by $i = 0..15$, the 15 AC transform coefficient levels of the i -th block are parsed into the i -th list `i16x16AClevel[ i ]`.

- Otherwise (`MbPartPredMode( mb_type, 0 )` is not equal to `Intra_16x16`), the following applies.
 - If `transform_size_8x8_flag` is equal to 0, for each of the 16 4x4 luma blocks indexed by $i = 0..15$, the 16 transform coefficient levels of the i -th block are parsed into the i -th list `level[ i ]`.
 - Otherwise (`transform_size_8x8_flag` is equal to 1), for each of the 4 8x8 luma blocks indexed by $i8x8 = 0..3$, the following applies.
 - If `entropy_coding_mode_flag` is equal to 0, first for each of the 4 4x4 luma blocks indexed by $i4x4 = 0..3$, the 16 transform coefficient levels of the $i4x4$ -th block are parsed into the $(i8x8 * 4 + i4x4)$ -th list `level[ i8x8 * 4 + i4x4 ]`. Then, the 64 transform coefficient levels of the $i8x8$ -th 8x8 luma block which are indexed by $4 * i + i4x4$, where $i = 0..15$ and $i4x4 = 0..3$, are derived as `level8x8[ i8x8 ][ 4 * i + i4x4 ] = level[ i8x8 * 4 + i4x4 ][ i ]`.
 NOTE – The 4x4 luma blocks with `luma4x4BlkIdx = i8x8 * 4 + i4x4` containing every fourth transform coefficient level of the corresponding $i8x8$ -th 8x8 luma block with offset $i4x4$ are assumed to represent spatial locations given by the inverse 4x4 luma block scanning process in subclause 6.4.3.
 - Otherwise (`entropy_coding_mode_flag` is equal to 1), the 64 transform coefficient levels of the $i8x8$ -th block are parsed into the $i8x8$ -th list `level8x8[ i8x8 ]`.

7.4.5.3.2 Residual block CAVLC semantics

The function `TotalCoeff( coeff_token )` that is used in subclause 7.3.5.3.2 returns the number of non-zero transform coefficient levels derived from `coeff_token`.

The function `TrailingOnes( coeff_token )` that is used in subclause 7.3.5.3.2 returns the trailing ones derived from `coeff_token`.

coeff_token specifies the total number of non-zero transform coefficient levels and the number of trailing one transform coefficient levels in a transform coefficient level scan. A trailing one transform coefficient level is one of up to three consecutive non-zero transform coefficient levels having an absolute value equal to 1 at the end of a scan of non-zero transform coefficient levels. The range of `coeff_token` is specified in subclause 9.2.1.

trailing_ones_sign_flag specifies the sign of a trailing one transform coefficient level as follows.

- If `trailing_ones_sign_flag` is equal to 0, the corresponding transform coefficient level is decoded as +1.
- Otherwise (`trailing_ones_sign_flag` equal to 1), the corresponding transform coefficient level is decoded as -1.

level_prefix and **level_suffix** specify the value of a non-zero transform coefficient level. The range of `level_prefix` and `level_suffix` is specified in subclause 9.2.2.

total_zeros specifies the total number of zero-valued transform coefficient levels that are located before the position of the last non-zero transform coefficient level in a scan of transform coefficient levels. The range of `total_zeros` is specified in subclause 9.2.3.

run_before specifies the number of consecutive transform coefficient levels in the scan with zero value before a non-zero valued transform coefficient level. The range of `run_before` is specified in subclause 9.2.3.

`coeffLevel` contains `maxNumCoeff` transform coefficient levels for the current list of transform coefficient levels.

7.4.5.3.3 Residual block CABAC semantics

coded_block_flag specifies whether the block contains non-zero transform coefficient levels as follows.

- If `coded_block_flag` is equal to 0, the block contains no non-zero transform coefficient levels.
- Otherwise (`coded_block_flag` is equal to 1), the block contains at least one non-zero transform coefficient level.

When `coded_block_flag` is not present, it is inferred to be equal to 1.

significant_coeff_flag[i] specifies whether the transform coefficient level at scanning position i is non-zero as follows.

- If `significant_coeff_flag[ i ]` is equal to 0, the transform coefficient level at scanning position i is set equal to 0;
- Otherwise (`significant_coeff_flag[ i ]` is equal to 1), the transform coefficient level at scanning position i has a non-zero value.

last_significant_coeff_flag[i] specifies for the scanning position *i* whether there are non-zero transform coefficient levels for subsequent scanning positions *i* + 1 to **maxNumCoeff** – 1 as follows.

- If **last_significant_coeff_flag[i]** is equal to 1, all following transform coefficient levels (in scanning order) of the block have value equal to 0.
- Otherwise (**last_significant_coeff_flag[i]** is equal to 0), there are further non-zero transform coefficient levels along the scanning path.

coeff_abs_level_minus1[i] is the absolute value of a transform coefficient level minus 1. The value of **coeff_abs_level_minus1** is constrained by the limits in subclause 8.5.

coeff_sign_flag[i] specifies the sign of a transform coefficient level as follows.

- If **coeff_sign_flag** is equal to 0, the corresponding transform coefficient level has a positive value.
- Otherwise (**coeff_sign_flag** is equal to 1), the corresponding transform coefficient level has a negative value.

coeffLevel contains **maxNumCoeff** transform coefficient levels for the current list of transform coefficient levels.

8 Decoding process

Outputs of this process are decoded samples of the current picture (sometimes referred to by the variable **CurrPic**).

Depending on the value of **chroma_format_idc**, the number of sample arrays of the current picture is as follows.

- If **chroma_format_idc** is equal to 0, the current picture consists of 1 sample array **S_L**.
- Otherwise (**chroma_format_idc** is not equal to 0), the current picture consists of 3 sample arrays **S_L**, **S_{Cb}**, **S_{Cr}**.

This clause describes the decoding process, given syntax elements and upper-case variables from clause 7.

The decoding process is specified such that all decoders shall produce numerically identical results. Any decoding process that produces identical results to the process described here conforms to the decoding process requirements of this Recommendation | International Standard.

Each picture referred to in this clause is a complete or part of a primary coded picture. Each slice referred to in this clause is a slice of a primary coded picture. Each slice data partition referred to in this clause is a slice data partition of a primary coded picture.

Depending on the value of **separate_colour_plane_flag**, the decoding process is structured as follows.

- If **separate_colour_plane_flag** is equal to 0, the decoding process is invoked a single time with the current picture being the output.
- Otherwise (**separate_colour_plane_flag** is equal to 1), the decoding process is invoked three times. Inputs to the decoding process are all NAL units of the primary coded picture with identical value of **colour_plane_id**. The decoding process of NAL units with a particular value of **colour_plane_id** is specified as if only a coded video sequence with monochrome colour format with that particular value of **colour_plane_id** would be present in the bitstream. The output of each of the three decoding processes is assigned to the 3 sample arrays of the current picture with the NAL units with **colour_plane_id** equal to 0 being assigned to **S_L**, the NAL units with **colour_plane_id** equal to 1 being assigned to **S_{Cb}**, and the NAL units with **colour_plane_id** equal to 2 being assigned to **S_{Cr}**.

NOTE - The variable **ChromaArrayType** is derived as 0 when **separate_colour_plane_flag** is equal to 1 and **chroma_format_idc** is equal to 3. In the decoding process, the value of this variable is evaluated resulting in operations identical to that of monochrome pictures with **chroma_format_idc** being equal to 0.

An overview of the decoding process is given as follows.

- The decoding of NAL units is specified in subclause 8.1.
- The processes in subclause 8.2 specify decoding processes using syntax elements in the slice layer and above.
 - Variables and functions relating to picture order count are derived in subclause 8.2.1. (only needed to be invoked for one slice of a picture).
 - Variables and functions relating to the macroblock to slice group map are derived in subclause 8.2.2. (only needed to be invoked for one slice of a picture).
 - The method of combining the various partitions when slice data partitioning is used is described in subclause 8.2.3.

- When the frame_num of the current picture is not equal to PrevRefFrameNum and is not equal to $(\text{PrevRefFrameNum} + 1) \% \text{MaxFrameNum}$, the decoding process for gaps in frame_num is performed according to subclause 8.2.5.2 prior to the decoding of any slices of the current picture.
- At the beginning of the decoding process for each P, SP, or B slice, the decoding process for reference picture lists construction specified in subclause 8.2.4 performed for derivation of reference picture list 0 (RefPicList0), and when decoding a B slice, reference picture list 1 (RefPicList1).
- When the current picture is a reference picture and after all slices of the current picture have been decoded, the decoded reference picture marking process in subclause 8.2.5 specifies how the current picture is used in the decoding process of inter prediction in later decoded pictures.
- The processes in subclauses 8.3, 8.4, 8.5, 8.6, and 8.7 specify decoding processes using syntax elements in the macroblock layer and above.
 - The intra prediction process for I and SI macroblocks, except for I_PCM macroblocks as specified in subclause 8.3, has intra prediction samples as its output. For I_PCM macroblocks subclause 8.3 directly specifies a picture construction process. The output are the constructed samples prior to the deblocking filter process.
 - The inter prediction process for P and B macroblocks is specified in subclause 8.4 with inter prediction samples being the output.
 - The transform coefficient decoding process and picture construction process prior to deblocking filter process are specified in subclause 8.5. That process derives samples for I and B macroblocks and for P macroblocks in P slices. The output are constructed samples prior to the deblocking filter process.
 - The decoding process for P macroblocks in SP slices or SI macroblocks is specified in subclause 8.6. That process derives samples for P macroblocks in SP slices and for SI macroblocks. The output are constructed samples prior to the deblocking filter process.
 - The constructed samples prior to the deblocking filter process that are next to the edges of blocks and macroblocks are processed by a deblocking filter as specified in subclause 8.7 with the output being the decoded samples.

8.1 NAL unit decoding process

Inputs to this process are NAL units.

Outputs of this process are the RBSP syntax structures encapsulated within the NAL units.

The decoding process for each NAL unit extracts the RBSP syntax structure from the NAL unit and then operates the decoding processes specified for the RBSP syntax structure in the NAL unit as follows.

Subclause 8.2 describes the decoding process for NAL units with nal_unit_type equal to 1 through 5.

Subclauses 8.3 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1, 2, and 5.

Subclause 8.4 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 2.

Subclause 8.5 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.6 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 and 3 to 5.

Subclause 8.7 describes the decoding process for a macroblock or part of a macroblock coded in NAL units with nal_unit_type equal to 1 to 5.

NAL units with nal_unit_type equal to 7 and 8 contain sequence parameter sets and picture parameter sets, respectively. Picture parameter sets are used in the decoding processes of other NAL units as determined by reference to a picture parameter set within the slice headers of each picture. Sequence parameter sets are used in the decoding processes of other NAL units as determined by reference to a sequence parameter set within the picture parameter sets of each sequence.

No normative decoding process is specified for NAL units with nal_unit_type equal to 6, 9, 10, 11, and 12.

8.2 Slice decoding process

8.2.1 Decoding process for picture order count

Outputs of this process are TopFieldOrderCnt (if applicable) and BottomFieldOrderCnt (if applicable).

Picture order counts are used to determine initial picture orderings for reference pictures in the decoding of B slices (see subclauses 8.2.4.2.3 and 8.2.4.2.4), to represent picture order differences between frames or fields for motion vector derivation in temporal direct mode (see subclause 8.4.1.2.3), for implicit mode weighted prediction in B slices (see subclause 8.4.2.3.2), and for decoder conformance checking (see subclause C.4).

Picture order count information is derived for every frame, field (whether decoded from a coded field or as a part of a decoded frame), or complementary field pair as follows:

- Each coded frame is associated with two picture order counts, called TopFieldOrderCnt and BottomFieldOrderCnt for its top field and bottom field, respectively.
- Each coded field is associated with a picture order count, called TopFieldOrderCnt for a coded top field and BottomFieldOrderCnt for a bottom field.
- Each complementary field pair is associated with two picture order counts, which are the TopFieldOrderCnt for its coded top field and the BottomFieldOrderCnt for its coded bottom field, respectively.

TopFieldOrderCnt and BottomFieldOrderCnt indicate the picture order of the corresponding top field or bottom field relative to the first output field of the previous IDR picture or the previous reference picture including a memory_management_control_operation equal to 5 in decoding order.

TopFieldOrderCnt and BottomFieldOrderCnt are derived by invoking one of the decoding processes for picture order count type 0, 1, and 2 in subclauses 8.2.1.1, 8.2.1.2, and 8.2.1.3, respectively. When the current picture includes a memory management control operation equal to 5, after the decoding of the current picture, tempPicOrderCnt is set equal to PicOrderCnt(CurrPic), TopFieldOrderCnt of the current picture (if any) is set equal to TopFieldOrderCnt - tempPicOrderCnt, and BottomFieldOrderCnt of the current picture (if any) is set equal to BottomFieldOrderCnt - tempPicOrderCnt.

The bitstream shall not contain data that results in $\text{Min}(\text{TopFieldOrderCnt}, \text{BottomFieldOrderCnt})$ not equal to 0 for a coded IDR frame, TopFieldOrderCnt not equal to 0 for a coded IDR top field, or BottomFieldOrderCnt not equal to 0 for a coded IDR bottom field. Thus, at least one of TopFieldOrderCnt and BottomFieldOrderCnt shall be equal to 0 for the fields of a coded IDR frame.

When the current picture is not an IDR picture, the following applies.

- Consider the list variable listD containing as elements the TopFieldOrderCnt and BottomFieldOrderCnt values associated with the list of pictures including all of the following
 - the first picture in the list is the previous picture of any of the following types
 - an IDR picture
 - a picture containing a memory_management_control_operation equal to 5
 - the following additional pictures.
 - If pic_order_cnt_type is equal to 0, all other pictures that follow in decoding order after the first picture in the list and are not "non-existing" frames inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2 and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is equal to 0 and the current picture is not a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.
 - Otherwise (pic_order_cnt_type is not equal to 0), all other pictures that follow in decoding order after the first picture in the list and either precede the current picture in decoding order or are the current picture. When pic_order_cnt_type is not equal to 0, the current picture is included in listD prior to the invoking of the decoded reference picture marking process.
- Consider the list variable listO which contains the elements of listD sorted in ascending order. listO shall not contain any of the following.
 - a pair of TopFieldOrderCnt and BottomFieldOrderCnt for a frame or complementary field pair that are not at consecutive positions in listO.
 - a TopFieldOrderCnt that has a value equal to another TopFieldOrderCnt.
 - a BottomFieldOrderCnt that has a value equal to another BottomFieldOrderCnt.

- a BottomFieldOrderCnt that has a value equal to a TopFieldOrderCnt unless the BottomFieldOrderCnt and TopFieldOrderCnt belong to the same coded frame or complementary field pair.

The bitstream shall not contain data that results in values of TopFieldOrderCnt, BottomFieldOrderCnt, PicOrderCntMsb, or FrameNumOffset used in the decoding process as specified in subclauses 8.2.1.1 to 8.2.1.3 that exceed the range of values from -2^{31} to $2^{31}-1$, inclusive.

The function PicOrderCnt(picX) is specified as follows:

```

if( picX is a frame or a complementary field pair )
    PicOrderCnt( picX ) = Min( TopFieldOrderCnt, BottomFieldOrderCnt ) of the frame or complementary field
pair picX
else if( picX is a top field )
    PicOrderCnt( picX ) = TopFieldOrderCnt of field picX
else if( picX is a bottom field )
    PicOrderCnt( picX ) = BottomFieldOrderCnt of field picX

```

(8-1)

Then DiffPicOrderCnt(picA, picB) is specified as follows:

$$\text{DiffPicOrderCnt}(\text{picA}, \text{picB}) = \text{PicOrderCnt}(\text{picA}) - \text{PicOrderCnt}(\text{picB})$$

(8-2)

The bitstream shall not contain data that results in values of DiffPicOrderCnt(picA, picB) used in the decoding process that exceed the range of -2^{15} to $2^{15} - 1$, inclusive.

NOTE 1 – Let X be the current picture and Y and Z be two other pictures in the same sequence, Y and Z are considered to be in the same output order direction from X when both DiffPicOrderCnt(X, Y) and DiffPicOrderCnt(X, Z) are positive or both are negative.

NOTE 2 – Many applications assign PicOrderCnt(X) proportional to the sampling time of the picture X relative to the sampling time of an IDR picture.

When the current picture includes a memory_management_control_operation equal to 5, PicOrderCnt(CurrPic) shall be greater than PicOrderCnt(any other picture in listD).

8.2.1.1 Decoding process for picture order count type 0

This process is invoked when pic_order_cnt_type is equal to 0.

Input to this process is PicOrderCntMsb of the previous reference picture in decoding order as specified in this subclause.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The variables prevPicOrderCntMsb and prevPicOrderCntLsb are derived as follows.

- If the current picture is an IDR picture, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.
- Otherwise (the current picture is not an IDR picture), the following applies.
 - If the previous reference picture in decoding order included a memory_management_control_operation equal to 5, the following applies.
 - If the previous reference picture in decoding order is not a bottom field, prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to the value of TopFieldOrderCnt for the previous reference picture in decoding order.
 - Otherwise (the previous reference picture in decoding order is a bottom field), prevPicOrderCntMsb is set equal to 0 and prevPicOrderCntLsb is set equal to 0.
 - Otherwise (the previous reference picture in decoding order did not include a memory_management_control_operation equal to 5), prevPicOrderCntMsb is set equal to PicOrderCntMsb of the previous reference picture in decoding order and prevPicOrderCntLsb is set equal to the value of pic_order_cnt_lsb of the previous reference picture in decoding order.

PicOrderCntMsb of the current picture is derived as follows:

```

if( ( pic_order_cnt_lsb < prevPicOrderCntLsb ) &&
    ( ( prevPicOrderCntLsb - pic_order_cnt_lsb ) >= ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb + MaxPicOrderCntLsb
else if( ( pic_order_cnt_lsb > prevPicOrderCntLsb ) &&
    ( ( pic_order_cnt_lsb - prevPicOrderCntLsb ) > ( MaxPicOrderCntLsb / 2 ) ) )
    PicOrderCntMsb = prevPicOrderCntMsb - MaxPicOrderCntLsb
else
    PicOrderCntMsb = prevPicOrderCntMsb

```

(8-3)

When the current picture is not a bottom field, TopFieldOrderCnt is derived as follows:

```

if( !field_pic_flag || !bottom_field_flag )
    TopFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb

```

(8-4)

When the current picture is not a top field, BottomFieldOrderCnt is derived as follows:

```

if( !field_pic_flag )
    BottomFieldOrderCnt = TopFieldOrderCnt + delta_pic_order_cnt_bottom
else if( bottom_field_flag )
    BottomFieldOrderCnt = PicOrderCntMsb + pic_order_cnt_lsb

```

(8-5)

8.2.1.2 Decoding process for picture order count type 1

This process is invoked when pic_order_cnt_type is equal to 1.

Input to this process is FrameNumOffset of the previous picture in decoding order as specified in this subclause.

Outputs of this process are either or both TopFieldOrderCnt or BottomFieldOrderCnt.

The values of TopFieldOrderCnt and BottomFieldOrderCnt are derived as specified in this subclause. Let prevFrameNum be equal to the frame_num of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable prevFrameNumOffset is derived as follows.

- If the previous picture in decoding order included a memory_management_control_operation equal to 5, prevFrameNumOffset is set equal to 0.
- Otherwise (the previous picture in decoding order did not include a memory_management_control_operation equal to 5), prevFrameNumOffset is set equal to the value of FrameNumOffset of the previous picture in decoding order.

NOTE – When gaps_in_frame_num_value_allowed_flag is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in frame_num specified in subclause 8.2.5.2.

The derivation proceeds in the following ordered steps.

1. The variable FrameNumOffset is derived as follows:

```

if( IdrPicFlag == 1 )
    FrameNumOffset = 0
else if( prevFrameNum > frame_num )
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset

```

(8-6)

2. The variable absFrameNum is derived as follows:

```

if( num_ref_frames_in_pic_order_cnt_cycle != 0 )
    absFrameNum = FrameNumOffset + frame_num
else
    absFrameNum = 0
if( nal_ref_idc == 0 && absFrameNum > 0 )
    absFrameNum = absFrameNum - 1

```

(8-7)

3. When `absFrameNum > 0`, `picOrderCntCycleCnt` and `frameNumInPicOrderCntCycle` are derived as follows:

```

if( absFrameNum > 0 ) {
    picOrderCntCycleCnt = ( absFrameNum - 1 ) / num_ref_frames_in_pic_order_cnt_cycle
    frameNumInPicOrderCntCycle = ( absFrameNum - 1 ) % num_ref_frames_in_pic_order_cnt_cycle
}

```

(8-8)

4. The variable `expectedDeltaPerPicOrderCntCycle` is derived as follows:

```

expectedDeltaPerPicOrderCntCycle = 0
for( i = 0; i < num_ref_frames_in_pic_order_cnt_cycle; i++ )
    expectedDeltaPerPicOrderCntCycle += offset_for_ref_frame[ i ]

```

(8-9)

5. The variable `expectedPicOrderCnt` is derived as follows:

```

if( absFrameNum > 0 ){
    expectedPicOrderCnt = picOrderCntCycleCnt * expectedDeltaPerPicOrderCntCycle
    for( i = 0; i <= frameNumInPicOrderCntCycle; i++ )
        expectedPicOrderCnt = expectedPicOrderCnt + offset_for_ref_frame[ i ]
} else
    expectedPicOrderCnt = 0
if( nal_ref_idc == 0 )
    expectedPicOrderCnt = expectedPicOrderCnt + offset_for_non_ref_pic

```

(8-10)

6. The variables `TopFieldOrderCnt` or `BottomFieldOrderCnt` are derived as follows:

```

if( !field_pic_flag ) {
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
    BottomFieldOrderCnt = TopFieldOrderCnt +
        offset_for_top_to_bottom_field + delta_pic_order_cnt[ 1 ]
} else if( !bottom_field_flag )
    TopFieldOrderCnt = expectedPicOrderCnt + delta_pic_order_cnt[ 0 ]
else
    BottomFieldOrderCnt = expectedPicOrderCnt + offset_for_top_to_bottom_field + delta_pic_order_cnt[ 0 ]

```

(8-11)

8.2.1.3 Decoding process for picture order count type 2

This process is invoked when `pic_order_cnt_type` is equal to 2.

Outputs of this process are either or both `TopFieldOrderCnt` or `BottomFieldOrderCnt`.

Let `prevFrameNum` be equal to the `frame_num` of the previous picture in decoding order.

When the current picture is not an IDR picture, the variable `prevFrameNumOffset` is derived as follows.

- If the previous picture in decoding order included a `memory_management_control_operation` equal to 5, `prevFrameNumOffset` is set equal to 0.
 - Otherwise (the previous picture in decoding order did not include a `memory_management_control_operation` equal to 5), `prevFrameNumOffset` is set equal to the value of `FrameNumOffset` of the previous picture in decoding order.
- NOTE 1 – When `gaps_in_frame_num_value_allowed_flag` is equal to 1, the previous picture in decoding order may be a "non-existing" frame inferred by the decoding process for gaps in `frame_num` specified in subclause 8.2.5.2.

The variable `FrameNumOffset` is derived as follows.

```

if( IdRPicFlag == 1 )
    FrameNumOffset = 0
else if( prevFrameNum > frame_num )
    FrameNumOffset = prevFrameNumOffset + MaxFrameNum
else
    FrameNumOffset = prevFrameNumOffset

```

(8-12)

The variable tempPicOrderCnt is derived as follows:

```

if( IdrPicFlag == 1 )
    tempPicOrderCnt = 0
else if( nal_ref_idc == 0 )
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num ) - 1
else
    tempPicOrderCnt = 2 * ( FrameNumOffset + frame_num )

```

(8-13)

The variables TopFieldOrderCnt or BottomFieldOrderCnt are derived as follows:

```

if( !field_pic_flag ) {
    TopFieldOrderCnt = tempPicOrderCnt
    BottomFieldOrderCnt = tempPicOrderCnt
} else if( bottom_field_flag )
    BottomFieldOrderCnt = tempPicOrderCnt
else
    TopFieldOrderCnt = tempPicOrderCnt

```

(8-14)

NOTE 2 – Picture order count type 2 cannot be used in a coded video sequence that contains consecutive non-reference pictures that would result in more than one of these pictures having the same value of TopFieldOrderCnt or more than one of these pictures having the same value of BottomFieldOrderCnt.

NOTE 3 – Picture order count type 2 results in an output order that is the same as the decoding order.

8.2.2 Decoding process for macroblock to slice group map

Inputs to this process are the active picture parameter set and the slice header of the slice to be decoded.

Output of this process is a macroblock to slice group map MbToSliceGroupMap.

This process is invoked at the start of every slice.

NOTE – The output of this process is equal for all slices of a picture.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 3, 4, or 5, slice groups 0 and 1 have a size and shape determined by slice_group_change_direction_flag as shown in Table 8-1 and specified in subclauses 8.2.2.4 to 8.2.2.6.

Table 8-1 – Refined slice group map type

slice_group_map_type	slice_group_change_direction_flag	refined slice group map type
3	0	Box-out clockwise
3	1	Box-out counter-clockwise
4	0	Raster scan
4	1	Reverse raster scan
5	0	Wipe right
5	1	Wipe left

In such a case, MapUnitsInSliceGroup0 slice group map units in the specified growth order are allocated for slice group 0 and the remaining PicSizeInMapUnits – MapUnitsInSliceGroup0 slice group map units of the picture are allocated for slice group 1.

When num_slice_groups_minus1 is equal to 1 and slice_group_map_type is equal to 4 or 5, the variable sizeOfUpperLeftGroup is defined as follows:

$$\text{sizeOfUpperLeftGroup} = (\text{slice_group_change_direction_flag} ? (\text{PicSizeInMapUnits} - \text{MapUnitsInSliceGroup0}) : \text{MapUnitsInSliceGroup0}) \quad (8-15)$$

The variable mapUnitToSliceGroupMap is derived as follows.

- If num_slice_groups_minus1 is equal to 0, the map unit to slice group map is generated for all i ranging from 0 to PicSizeInMapUnits – 1, inclusive, as specified by:

$$\text{mapUnitToSliceGroupMap}[i] = 0 \quad (8-16)$$

- Otherwise (num_slice_groups_minus1 is not equal to 0), mapUnitToSliceGroupMap is derived as follows.
 - If slice_group_map_type is equal to 0, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.1 applies.
 - Otherwise, if slice_group_map_type is equal to 1, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.2 applies.
 - Otherwise, if slice_group_map_type is equal to 2, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.3 applies.
 - Otherwise, if slice_group_map_type is equal to 3, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.4 applies.
 - Otherwise, if slice_group_map_type is equal to 4, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.5 applies.
 - Otherwise, if slice_group_map_type is equal to 5, the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.6 applies.
 - Otherwise (slice_group_map_type is equal to 6), the derivation of mapUnitToSliceGroupMap as specified in subclause 8.2.2.7 applies.

After derivation of the mapUnitToSliceGroupMap, the process specified in subclause 8.2.2.8 is invoked to convert the map unit to slice group map mapUnitToSliceGroupMap to the macroblock to slice group map MbToSliceGroupMap. After derivation of the macroblock to slice group map as specified in subclause 8.2.2.8, the function NextMbAddress(n) is defined as the value of the variable nextMbAddress derived as specified by:

$$\begin{aligned} i &= n + 1 \\ \text{while}(i < \text{PicSizeInMbs} \ \&\& \ \text{MbToSliceGroupMap}[i] \neq \text{MbToSliceGroupMap}[n]) \\ & \quad i++; \\ \text{nextMbAddress} &= i \end{aligned} \quad (8-17)$$

8.2.2.1 Specification for interleaved slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 0.

The map unit to slice group map is generated as specified by:

$$\begin{aligned} i &= 0 \\ \text{do} \\ & \quad \text{for}(iGroup = 0; iGroup \leq \text{num_slice_groups_minus1} \ \&\& \ i < \text{PicSizeInMapUnits}; \\ & \quad \quad i += \text{run_length_minus1}[iGroup++] + 1) \\ & \quad \quad \text{for}(j = 0; j \leq \text{run_length_minus1}[iGroup] \ \&\& \ i + j < \text{PicSizeInMapUnits}; j++) \\ & \quad \quad \quad \text{mapUnitToSliceGroupMap}[i + j] = iGroup \\ \text{while}(i < \text{PicSizeInMapUnits}) \end{aligned} \quad (8-18)$$

8.2.2.2 Specification for dispersed slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 1.

The map unit to slice group map is generated as specified by:

$$\begin{aligned} \text{for}(i = 0; i < \text{PicSizeInMapUnits}; i++) \\ \text{mapUnitToSliceGroupMap}[i] &= ((i \% \text{PicWidthInMbs}) + \\ & \quad ((i / \text{PicWidthInMbs}) * (\text{num_slice_groups_minus1} + 1)) / 2) \\ & \quad \% (\text{num_slice_groups_minus1} + 1) \end{aligned} \quad (8-19)$$

8.2.2.3 Specification for foreground with left-over slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 2.

The map unit to slice group map is generated as specified by:

```

for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = num_slice_groups_minus1
for( iGroup = num_slice_groups_minus1 - 1; iGroup >= 0; iGroup-- ) {
    yTopLeft = top_left[ iGroup ] / PicWidthInMbs
    xTopLeft = top_left[ iGroup ] % PicWidthInMbs
    yBottomRight = bottom_right[ iGroup ] / PicWidthInMbs
    xBottomRight = bottom_right[ iGroup ] % PicWidthInMbs
    for( y = yTopLeft; y <= yBottomRight; y++ )
        for( x = xTopLeft; x <= xBottomRight; x++ )
            mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = iGroup
}

```

(8-20)

NOTE – The rectangles may overlap. Slice group 0 contains the macroblocks that are within the rectangle specified by `top_left[ 0 ]` and `bottom_right[ 0 ]`. A slice group having slice group ID greater than 0 and less than `num_slice_groups_minus1` contains the macroblocks that are within the specified rectangle for that slice group that are not within the rectangle specified for any slice group having a smaller slice group ID. The slice group with slice group ID equal to `num_slice_groups_minus1` contains the macroblocks that are not in the other slice groups.

8.2.2.4 Specification for box-out slice group map types

The specifications in this subclause apply when `slice_group_map_type` is equal to 3.

The map unit to slice group map is generated as specified by:

```

for( i = 0; i < PicSizeInMapUnits; i++ )
    mapUnitToSliceGroupMap[ i ] = 1
x = ( PicWidthInMbs - slice_group_change_direction_flag ) / 2
y = ( PicHeightInMapUnits - slice_group_change_direction_flag ) / 2
( leftBound, topBound ) = ( x, y )
( rightBound, bottomBound ) = ( x, y )
( xDir, yDir ) = ( slice_group_change_direction_flag - 1, slice_group_change_direction_flag )
for( k = 0; k < MapUnitsInSliceGroup0; k += mapUnitVacant ) {
    mapUnitVacant = ( mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] == 1 )
    if( mapUnitVacant )
        mapUnitToSliceGroupMap[ y * PicWidthInMbs + x ] = 0
    if( xDir == -1 && x == leftBound ) {
        leftBound = Max( leftBound - 1, 0 )
        x = leftBound
        ( xDir, yDir ) = ( 0, 2 * slice_group_change_direction_flag - 1 )
    } else if( xDir == 1 && x == rightBound ) {
        rightBound = Min( rightBound + 1, PicWidthInMbs - 1 )
        x = rightBound
        ( xDir, yDir ) = ( 0, 1 - 2 * slice_group_change_direction_flag )
    } else if( yDir == -1 && y == topBound ) {
        topBound = Max( topBound - 1, 0 )
        y = topBound
        ( xDir, yDir ) = ( 1 - 2 * slice_group_change_direction_flag, 0 )
    } else if( yDir == 1 && y == bottomBound ) {
        bottomBound = Min( bottomBound + 1, PicHeightInMapUnits - 1 )
        y = bottomBound
        ( xDir, yDir ) = ( 2 * slice_group_change_direction_flag - 1, 0 )
    } else
        ( x, y ) = ( x + xDir, y + yDir )
}

```

(8-21)

8.2.2.5 Specification for raster scan slice group map types

The specifications in this subclause apply when `slice_group_map_type` is equal to 4.

The map unit to slice group map is generated as specified by:

```

for( i = 0; i < PicSizeInMapUnits; i++ )
    if( i < sizeOfUpperLeftGroup )
        mapUnitToSliceGroupMap[ i ] = slice_group_change_direction_flag
    else
        mapUnitToSliceGroupMap[ i ] = 1 - slice_group_change_direction_flag

```

(8-22)

8.2.2.6 Specification for wipe slice group map types

The specifications in this subclause apply when slice_group_map_type is equal to 5.

The map unit to slice group map is generated as specified by:

```

k = 0;
for( j = 0; j < PicWidthInMbs; j++ )
    for( i = 0; i < PicHeightInMapUnits; i++ )
        if( k++ < sizeOfUpperLeftGroup )
            mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = slice_group_change_direction_flag
        else
            mapUnitToSliceGroupMap[ i * PicWidthInMbs + j ] = 1 - slice_group_change_direction_flag

```

(8-23)

8.2.2.7 Specification for explicit slice group map type

The specifications in this subclause apply when slice_group_map_type is equal to 6.

The map unit to slice group map is generated as specified by:

$$\text{mapUnitToSliceGroupMap}[i] = \text{slice_group_id}[i] \quad (8-24)$$

for all i ranging from 0 to $\text{PicSizeInMapUnits} - 1$, inclusive.

8.2.2.8 Specification for conversion of map unit to slice group map to macroblock to slice group map

For each value of i ranging from 0 to $\text{PicSizeInMbs} - 1$, inclusive, the macroblock to slice group map is specified as follows.

- If frame_mbs_only_flag is equal to 1 or field_pic_flag is equal to 1, the macroblock to slice group map is specified by:

$$\text{MbToSliceGroupMap}[i] = \text{mapUnitToSliceGroupMap}[i] \quad (8-25)$$

- Otherwise, if MbaffFrameFlag is equal to 1, the macroblock to slice group map is specified by:

$$\text{MbToSliceGroupMap}[i] = \text{mapUnitToSliceGroupMap}[i / 2] \quad (8-26)$$

- Otherwise (frame_mbs_only_flag is equal to 0 and mb_adaptive_frame_field_flag is equal to 0 and field_pic_flag is equal to 0), the macroblock to slice group map is specified by:

$$\text{MbToSliceGroupMap}[i] = \text{mapUnitToSliceGroupMap}[(i / (2 * \text{PicWidthInMbs})) * \text{PicWidthInMbs} + (i \% \text{PicWidthInMbs})] \quad (8-27)$$

8.2.3 Decoding process for slice data partitioning

Inputs to this process are

- a slice data partition A layer RBSP,
- when syntax elements of category 3 are present in the slice data, a slice data partition B layer RBSP having the same slice_id as in the slice data partition A layer RBSP, and
- when syntax elements of category 4 are present in the slice data, a slice data partition C layer RBSP having the same slice_id as in the slice data partition A layer RBSP.

NOTE 1 – The slice data partition B layer RBSP and slice data partition C layer RBSP need not be present.

Output of this process is a coded slice.