
**Information technology — High efficiency
coding and media delivery in
heterogeneous environments**

**Part 2:
High efficiency video coding**

*Technologies de l'information — Codage à haute efficacité et livraison
des médias dans des environnements hétérogènes —*

Partie 2: Codage vidéo à haute efficacité



Reference number
ISO/IEC 23008-2:2015(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2015



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2015

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

CONTENTS

Page

0	Introduction.....	xvii
0.1	General.....	xvii
0.2	Prologue.....	xvii
0.3	Purpose.....	xvii
0.4	Applications.....	xvii
0.5	Publication and versions of this Specification.....	xvii
0.6	Profiles, tiers, and levels.....	xviii
0.7	Overview of the design characteristics.....	xviii
0.8	How to read this Specification.....	xviii
1	Scope.....	1
2	Normative references.....	1
2.1	General.....	1
2.2	Identical Recommendations International Standards.....	1
2.3	Paired Recommendations International Standards equivalent in technical content.....	1
2.4	Additional references.....	1
3	Definitions.....	1
4	Abbreviations.....	10
5	Conventions.....	12
5.1	General.....	12
5.2	Arithmetic operators.....	12
5.3	Logical operators.....	12
5.4	Relational operators.....	12
5.5	Bit-wise operators.....	12
5.6	Assignment operators.....	13
5.7	Range notation.....	13
5.8	Mathematical functions.....	13
5.9	Order of operation precedence.....	14
5.10	Variables, syntax elements, and tables.....	15
5.11	Text description of logical operations.....	15
5.12	Processes.....	16
6	Bitstream and picture formats, partitionings, scanning processes, and neighbouring relationships.....	17
6.1	Bitstream formats.....	17
6.2	Source, decoded, and output picture formats.....	17
6.3	Partitioning of pictures, slices, slice segments, tiles, coding tree units, and coding tree blocks.....	19
6.3.1	Partitioning of pictures into slices, slice segments, and tiles.....	19
6.3.2	Block and quadtree structures.....	20
6.3.3	Spatial or component-wise partitionings.....	21
6.4	Availability processes.....	22
6.4.1	Derivation process for z-scan order block availability.....	22
6.4.2	Derivation process for prediction block availability.....	22
6.5	Scanning processes.....	23
6.5.1	Coding tree block raster and tile scanning conversion process.....	23
6.5.2	Z-scan order array initialization process.....	24
6.5.3	Up-right diagonal scan order array initialization process.....	25
6.5.4	Horizontal scan order array initialization process.....	25
6.5.5	Vertical scan order array initialization process.....	25
7	Syntax and semantics.....	26
7.1	Method of specifying syntax in tabular form.....	26
7.2	Specification of syntax functions and descriptors.....	27
7.3	Syntax in tabular form.....	28
7.3.1	NAL unit syntax.....	28

7.3.1.1	General NAL unit syntax	28
7.3.1.2	NAL unit header syntax	29
7.3.2	Raw byte sequence payloads, trailing bits, and byte alignment syntax	30
7.3.2.1	Video parameter set RBSP syntax	30
7.3.2.2	Sequence parameter set RBSP syntax	31
7.3.2.3	Picture parameter set RBSP syntax	34
7.3.2.4	Supplemental enhancement information RBSP syntax	36
7.3.2.5	Access unit delimiter RBSP syntax	36
7.3.2.6	End of sequence RBSP syntax	36
7.3.2.7	End of bitstream RBSP syntax	36
7.3.2.8	Filler data RBSP syntax	36
7.3.2.9	Slice segment layer RBSP syntax	36
7.3.2.10	RBSP slice segment trailing bits syntax	37
7.3.2.11	RBSP trailing bits syntax	37
7.3.2.12	Byte alignment syntax	37
7.3.3	Profile, tier and level syntax	38
7.3.4	Scaling list data syntax	40
7.3.5	Supplemental enhancement information message syntax	40
7.3.6	Slice segment header syntax	41
7.3.6.1	General slice segment header syntax	41
7.3.6.2	Reference picture list modification syntax	43
7.3.6.3	Weighted prediction parameters syntax	44
7.3.7	Short-term reference picture set syntax	45
7.3.8	Slice segment data syntax	46
7.3.8.1	General slice segment data syntax	46
7.3.8.2	Coding tree unit syntax	46
7.3.8.3	Sample adaptive offset syntax	46
7.3.8.4	Coding quadtree syntax	48
7.3.8.5	Coding unit syntax	49
7.3.8.6	Prediction unit syntax	51
7.3.8.7	PCM sample syntax	51
7.3.8.8	Transform tree syntax	52
7.3.8.9	Motion vector difference syntax	53
7.3.8.10	Transform unit syntax	54
7.3.8.11	Residual coding syntax	56
7.3.8.12	Cross-component prediction syntax	58
7.4	Semantics	58
7.4.1	General	58
7.4.2	NAL unit semantics	58
7.4.2.1	General NAL unit semantics	58
7.4.2.2	NAL unit header semantics	59
7.4.2.3	Encapsulation of an SODB within an RBSP (informative)	62
7.4.2.4	Order of NAL units and association to coded pictures, access units, and coded video sequences	63
7.4.3	Raw byte sequence payloads, trailing bits, and byte alignment semantics	66
7.4.3.1	Video parameter set RBSP semantics	66
7.4.3.2	Sequence parameter set RBSP semantics	69
7.4.3.3	Picture parameter set RBSP semantics	75
7.4.3.4	Supplemental enhancement information RBSP semantics	79
7.4.3.5	Access unit delimiter RBSP semantics	79
7.4.3.6	End of sequence RBSP semantics	79
7.4.3.7	End of bitstream RBSP semantics	80
7.4.3.8	Filler data RBSP semantics	80
7.4.3.9	Slice segment layer RBSP semantics	80
7.4.3.10	RBSP slice segment trailing bits semantics	80
7.4.3.11	RBSP trailing bits semantics	80
7.4.3.12	Byte alignment semantics	80
7.4.4	Profile, tier, and level semantics	80
7.4.5	Scaling list data semantics	83
7.4.6	Supplemental enhancement information message semantics	85
7.4.7	Slice segment header semantics	85
7.4.7.1	General slice segment header semantics	85
7.4.7.2	Reference picture list modification semantics	90
7.4.7.3	Weighted prediction parameters semantics	91

7.4.8	Short-term reference picture set semantics	91
7.4.9	Slice segment data semantics	94
7.4.9.1	General slice segment data semantics	94
7.4.9.2	Coding tree unit semantics	94
7.4.9.3	Sample adaptive offset semantics	94
7.4.9.4	Coding quadtree semantics	96
7.4.9.5	Coding unit semantics	96
7.4.9.6	Prediction unit semantics	98
7.4.9.7	PCM sample semantics	98
7.4.9.8	Transform tree semantics	99
7.4.9.9	Motion vector difference semantics	99
7.4.9.10	Transform unit semantics	100
7.4.9.11	Residual coding semantics	101
7.4.9.12	Cross-component prediction semantics	103
8	Decoding process	103
8.1	General decoding process	103
8.1.1	General	103
8.1.2	CVSG decoding process	104
8.1.3	Decoding process for a coded picture with nuh_layer_id equal to 0	104
8.2	NAL unit decoding process	105
8.3	Slice decoding process	105
8.3.1	Decoding process for picture order count	105
8.3.2	Decoding process for reference picture set	106
8.3.3	Decoding process for generating unavailable reference pictures	110
8.3.3.1	General decoding process for generating unavailable reference pictures	110
8.3.3.2	Generation of one unavailable picture	111
8.3.4	Decoding process for reference picture lists construction	111
8.3.5	Decoding process for collocated picture and no backward prediction flag	112
8.4	Decoding process for coding units coded in intra prediction mode	112
8.4.1	General decoding process for coding units coded in intra prediction mode	112
8.4.2	Derivation process for luma intra prediction mode	113
8.4.3	Derivation process for chroma intra prediction mode	115
8.4.4	Decoding process for intra blocks	116
8.4.4.1	General decoding process for intra blocks	116
8.4.4.2	Intra sample prediction	118
8.5	Decoding process for coding units coded in inter prediction mode	124
8.5.1	General decoding process for coding units coded in inter prediction mode	124
8.5.2	Inter prediction process	124
8.5.3	Decoding process for prediction units in inter prediction mode	127
8.5.3.1	General	127
8.5.3.2	Derivation process for motion vector components and reference indices	128
8.5.3.3	Decoding process for inter prediction samples	142
8.5.4	Decoding process for the residual signal of coding units coded in inter prediction mode	150
8.5.4.1	General	150
8.5.4.2	Decoding process for luma residual blocks	151
8.5.4.3	Decoding process for chroma residual blocks	152
8.6	Scaling, transformation and array construction process prior to deblocking filter process	153
8.6.1	Derivation process for quantization parameters	153
8.6.2	Scaling and transformation process	155
8.6.3	Scaling process for transform coefficients	156
8.6.4	Transformation process for scaled transform coefficients	157
8.6.4.1	General	157
8.6.4.2	Transformation process	158
8.6.5	Residual modification process for blocks using a transform bypass	159
8.6.6	Residual modification process for transform blocks using cross-component prediction	159
8.6.7	Picture construction process prior to in-loop filter process	160
8.7	In-loop filter process	160
8.7.1	General	160
8.7.2	Deblocking filter process	161
8.7.2.1	General	161
8.7.2.2	Derivation process of transform block boundary	162
8.7.2.3	Derivation process of prediction block boundary	163

8.7.2.4	Derivation process of boundary filtering strength.....	164
8.7.2.5	Edge filtering process	165
8.7.3	Sample adaptive offset process	173
8.7.3.1	General.....	173
8.7.3.2	Coding tree block modification process.....	174
9	Parsing process	175
9.1	General.....	175
9.2	Parsing process for 0-th order Exp-Golomb codes	176
9.2.1	General.....	176
9.2.2	Mapping process for signed Exp-Golomb codes	177
9.3	CABAC parsing process for slice segment data	178
9.3.1	General.....	178
9.3.2	Initialization process	180
9.3.2.1	General.....	180
9.3.2.2	Initialization process for context variables	181
9.3.2.3	Storage process for context variables and Rice parameter initialization states.....	190
9.3.2.4	Synchronization process for context variables and Rice parameter initialization states	191
9.3.2.5	Initialization process for the arithmetic decoding engine	191
9.3.3	Binarization process.....	192
9.3.3.1	General.....	192
9.3.3.2	Truncated Rice (TR) binarization process	194
9.3.3.3	k-th order Exp-Golomb (EGk) binarization process	195
9.3.3.4	Limited k-th order Exp-Golomb (EGk) binarization process	195
9.3.3.5	Fixed-length (FL) binarization process.....	196
9.3.3.6	Binarization process for part_mode	196
9.3.3.7	Binarization process for intra_chroma_pred_mode	197
9.3.3.8	Binarization process for inter_pred_idc.....	197
9.3.3.9	Binarization process for cu_qp_delta_abs	198
9.3.3.10	Binarization process for coeff_abs_level_remaining[]	198
9.3.4	Decoding process flow.....	199
9.3.4.1	General.....	199
9.3.4.2	Derivation process for ctxTable, ctxIdx and bypassFlag	200
9.3.4.3	Arithmetic decoding process.....	206
9.3.5	Arithmetic encoding process (informative).....	212
9.3.5.1	General.....	212
9.3.5.2	Initialization process for the arithmetic encoding engine (informative)	212
9.3.5.3	Encoding process for a binary decision (informative)	213
9.3.5.4	Renormalization process in the arithmetic encoding engine (informative).....	214
9.3.5.5	Bypass encoding process for binary decisions (informative).....	215
9.3.5.6	Encoding process for a binary decision before termination (informative).....	216
9.3.5.7	Byte stuffing process (informative)	218
10	Sub-bitstream extraction process	218
Annex A	Profiles, tiers and levels	219
A.1	Overview of profiles, tiers and levels	219
A.2	Requirements on video decoder capability	219
A.3	Profiles.....	219
A.3.1	General.....	219
A.3.2	Main profile	219
A.3.3	Main 10 profile	220
A.3.4	Main Still Picture profile.....	221
A.3.5	Format range extensions profiles	222
A.3.6	Format range extensions high throughput profiles	226
A.4	Tiers and levels	228
A.4.1	General tier and level limits	228
A.4.2	Profile-specific level limits for the video profiles.....	229
A.4.3	Effect of level limits on picture rate for the video profiles (informative)	232
Annex B	Byte stream format	236
B.1	General.....	236
B.2	Byte stream NAL unit syntax and semantics	236
B.2.1	Byte stream NAL unit syntax.....	236

B.2.2	Byte stream NAL unit semantics	236
B.3	Byte stream NAL unit decoding process	237
B.4	Decoder byte-alignment recovery (informative)	237
Annex C	Hypothetical reference decoder	238
C.1	General	238
C.2	Operation of coded picture buffer (CPB)	242
C.2.1	General	242
C.2.2	Timing of decoding unit arrival	242
C.2.3	Timing of decoding unit removal and decoding of decoding unit	244
C.3	Operation of the decoded picture buffer (DPB)	246
C.3.1	General	246
C.3.2	Removal of pictures from the DPB	247
C.3.3	Picture output	247
C.3.4	Current decoded picture marking and storage	248
C.4	Bitstream conformance	248
C.5	Decoder conformance	249
C.5.1	General	249
C.5.2	Operation of the output order DPB	250
C.5.2.1	General	250
C.5.2.2	Output and removal of pictures from the DPB	251
C.5.2.3	Picture decoding, marking, additional bumping, and storage	251
C.5.2.4	"Bumping" process	252
Annex D	Supplemental enhancement information	253
D.1	General	253
D.2	SEI payload syntax	254
D.2.1	General SEI message syntax	254
D.2.2	Buffering period SEI message syntax	257
D.2.3	Picture timing SEI message syntax	258
D.2.4	Pan-scan rectangle SEI message syntax	258
D.2.5	Filler payload SEI message syntax	259
D.2.6	User data registered by Rec. ITU-T T.35 SEI message syntax	259
D.2.7	User data unregistered SEI message syntax	259
D.2.8	Recovery point SEI message syntax	259
D.2.9	Scene information SEI message syntax	260
D.2.10	Picture snapshot SEI message syntax	260
D.2.11	Progressive refinement segment start SEI message syntax	260
D.2.12	Progressive refinement segment end SEI message syntax	260
D.2.13	Film grain characteristics SEI message syntax	261
D.2.14	Post-filter hint SEI message syntax	261
D.2.15	Tone mapping information SEI message syntax	262
D.2.16	Frame packing arrangement SEI message syntax	263
D.2.17	Display orientation SEI message syntax	263
D.2.18	Structure of pictures information SEI message syntax	264
D.2.19	Decoded picture hash SEI message syntax	264
D.2.20	Active parameter sets SEI message syntax	264
D.2.21	Decoding unit information SEI message syntax	265
D.2.22	Temporal sub-layer zero index SEI message syntax	265
D.2.23	Scalable nesting SEI message syntax	265
D.2.24	Region refresh information SEI message syntax	266
D.2.25	No display SEI message syntax	266
D.2.26	Time code SEI message syntax	267
D.2.27	Mastering display colour volume SEI message syntax	268
D.2.28	Segmented rectangular frame packing arrangement SEI message syntax	268
D.2.29	Temporal motion-constrained tile sets SEI message syntax	269
D.2.30	Chroma resampling filter hint SEI message syntax	270
D.2.31	Knee function information SEI message syntax	270
D.2.32	Colour remapping information SEI message syntax	271
D.2.33	Deinterlaced field identification SEI message syntax	272
D.2.34	Reserved SEI message syntax	272
D.3	SEI payload semantics	272
D.3.1	General SEI payload semantics	272

D.3.2	Buffering period SEI message semantics	276
D.3.3	Picture timing SEI message semantics	278
D.3.4	Pan-scan rectangle SEI message semantics	283
D.3.5	Filler payload SEI message semantics	284
D.3.6	User data registered by Rec. ITU-T T.35 SEI message semantics	284
D.3.7	User data unregistered SEI message semantics	284
D.3.8	Recovery point SEI message semantics	285
D.3.9	Scene information SEI message semantics	286
D.3.10	Picture snapshot SEI message semantics	288
D.3.11	Progressive refinement segment start SEI message semantics	288
D.3.12	Progressive refinement segment end SEI message semantics	289
D.3.13	Film grain characteristics SEI message semantics	289
D.3.14	Post-filter hint SEI message semantics	294
D.3.15	Tone mapping information SEI message semantics	295
D.3.16	Frame packing arrangement SEI message semantics	299
D.3.17	Display orientation SEI message semantics	306
D.3.18	Structure of pictures information SEI message semantics	307
D.3.19	Decoded picture hash SEI message semantics	308
D.3.20	Active parameter sets SEI message semantics	309
D.3.21	Decoding unit information SEI message semantics	310
D.3.22	Temporal sub-layer zero index SEI message semantics	311
D.3.23	Scalable nesting SEI message semantics	312
D.3.24	Region refresh information SEI message semantics	313
D.3.25	No display SEI message semantics	314
D.3.26	Time code SEI message semantics	314
D.3.27	Mastering display colour volume SEI message semantics	317
D.3.28	Segmented rectangular frame packing arrangement SEI message semantics	318
D.3.29	Temporal motion-constrained tile sets SEI message semantics	320
D.3.30	Chroma resampling filter hint SEI message semantics	323
D.3.31	Knee function information SEI message semantics	332
D.3.32	Colour remapping information SEI message semantics	334
D.3.33	Deinterlaced field identification SEI message semantics	336
D.3.34	Reserved SEI message semantics	336
Annex E	Video usability information	337
E.1	General	337
E.2	VUI syntax	338
E.2.1	VUI parameters syntax	338
E.2.2	HRD parameters syntax	340
E.2.3	Sub-layer HRD parameters syntax	341
E.3	VUI semantics	341
E.3.1	VUI parameters semantics	341
E.3.2	HRD parameters semantics	354
E.3.3	Sub-layer HRD parameters semantics	357
Annex F	Common specifications for multi-layer extensions	359
F.1	Scope	359
F.2	Normative references	359
F.3	Definitions	359
F.4	Abbreviations	362
F.5	Conventions	362
F.6	Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships	362
F.7	Syntax and semantics	362
F.7.1	Method of specifying syntax in tabular form	362
F.7.2	Specification of syntax functions, categories, and descriptors	362
F.7.3	Syntax in tabular form	362
F.7.3.1	NAL unit syntax	362
F.7.3.2	Raw byte sequence payloads and RBSP trailing bits syntax	363
F.7.3.3	Profile, tier, and level syntax	377
F.7.3.4	Scaling list data syntax	377
F.7.3.5	Supplemental enhancement information message syntax	377
F.7.3.6	Slice segment header syntax	378
F.7.3.7	Short-term reference picture set syntax	381

F.7.3.8	Slice segment data syntax	381
F.7.4	Semantics	381
F.7.4.1	General	381
F.7.4.2	NAL unit semantics	381
F.7.4.3	Raw byte sequence payloads, trailing bits, and byte alignment semantics	385
F.7.4.4	Profile, tier, and level semantics	410
F.7.4.5	Scaling list data semantics	412
F.7.4.6	Supplemental enhancement information message semantics	412
F.7.4.7	Slice segment header semantics	412
F.7.4.8	Short-term reference picture set semantics	416
F.7.4.9	Slice segment data semantics	417
F.8	Decoding process	417
F.8.1	General decoding process	417
F.8.1.1	General	417
F.8.1.2	CVSG decoding process	418
F.8.1.3	Common decoding process for a coded picture	419
F.8.1.4	Decoding process for a coded picture with nuh_layer_id equal to 0	422
F.8.1.5	Decoding process for starting the decoding of a coded picture with nuh_layer_id greater than 0	422
F.8.1.6	Decoding process for ending the decoding of a coded picture with nuh_layer_id greater than 0	423
F.8.1.7	Decoding process for generating unavailable reference pictures for pictures first in decoding order within a layer	423
F.8.1.8	Initialization process for an external base layer picture	424
F.8.1.9	Decoding process for an external base layer picture	425
F.8.2	NAL unit decoding process	425
F.8.3	Slice decoding processes	425
F.8.3.1	Decoding process for picture order count	425
F.8.3.2	Decoding process for reference picture set	427
F.8.3.3	Decoding process for generating unavailable reference pictures	428
F.8.3.4	Decoding process for reference picture lists construction	428
F.8.4	Decoding process for coding units coded in intra prediction mode	428
F.8.5	Decoding process for coding units coded in inter prediction mode	429
F.8.6	Scaling, transformation and array construction process prior to deblocking filter process	429
F.8.7	In-loop filter process	429
F.9	Parsing process	429
F.10	Specification of bitstream subsets	429
F.10.1	Sub-bitstream extraction process	429
F.10.2	Independent non-base layer rewriting process	429
F.10.3	Sub-bitstream extraction process for additional layer sets	430
F.11	Profiles, tiers, and levels	430
F.11.1	Independent non-base layer decoding capability	430
F.11.2	Decoder capabilities	431
F.12	Byte stream format	432
F.13	Hypothetical reference decoder	432
F.13.1	General	432
F.13.2	Operation of bitstream partition buffer (BPB)	437
F.13.2.1	General	437
F.13.2.2	Timing of decoding unit arrival	437
F.13.2.3	Timing of decoding unit removal and decoding of decoding unit	439
F.13.3	Operation of decoded picture buffer (DPB)	442
F.13.3.1	General	442
F.13.3.2	Removal of pictures from the DPB	442
F.13.3.3	Picture output	443
F.13.3.4	Current decoded picture marking and storage	444
F.13.4	Bitstream conformance	444
F.13.5	Decoder conformance	446
F.13.5.1	General	446
F.13.5.2	Operation of the output order DPB	447
F.13.6	Demultiplexing process for deriving a bitstream partition	450
F.14	Supplemental enhancement information	450
F.14.1	General	450
F.14.2	SEI payload syntax	450
F.14.2.1	General SEI payload syntax	450

F.14.2.2	Annex D SEI message syntax for multi-layer extensions	450
F.14.2.3	Layers not present SEI message syntax	450
F.14.2.4	Inter-layer constrained tile sets SEI message syntax	451
F.14.2.5	Bitstream partition nesting SEI message syntax	451
F.14.2.6	Bitstream partition initial arrival time SEI message syntax	452
F.14.2.7	Sub-bitstream property SEI message syntax	452
F.14.2.8	Alpha channel information SEI message syntax	452
F.14.2.9	Overlay information SEI message syntax	453
F.14.2.10	Temporal motion vector prediction constraints SEI message syntax	454
F.14.2.11	Frame-field information SEI message syntax	454
F.14.3	SEI payload semantics	454
F.14.3.1	General SEI payload semantics	454
F.14.3.2	Annex D SEI message semantics for multi-layer extensions	455
F.14.3.3	Layers not present SEI message semantics	463
F.14.3.4	Inter-layer constrained tile sets SEI message semantics	463
F.14.3.5	Bitstream partition nesting SEI message semantics	465
F.14.3.6	Bitstream partition initial arrival time SEI message semantics	465
F.14.3.7	Sub-bitstream property SEI message semantics	466
F.14.3.8	Alpha channel information SEI message semantics	467
F.14.3.9	Overlay information SEI message semantics	468
F.14.3.10	Temporal motion vector prediction constraints SEI message semantics	470
F.14.3.11	Frame-field information SEI message semantics	471
F.15	Video usability information	472
F.15.1	General	472
F.15.2	VUI syntax	472
F.15.3	VUI semantics	472
F.15.3.1	VUI parameters semantics	472
F.15.3.2	HRD parameters semantics	473
F.15.3.3	Sub-layer HRD parameters semantics	473
Annex G	Multiview high efficiency video coding	474
G.1	Scope	474
G.2	Normative references	474
G.3	Definitions	474
G.4	Abbreviations	474
G.5	Conventions	474
G.6	Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships	474
G.7	Syntax and semantics	474
G.8	Decoding processes	474
G.8.1	General decoding process	474
G.8.1.1	General	474
G.8.1.2	Decoding process for a coded picture with nuh_layer_id greater than 0	474
G.8.1.3	Decoding process for inter-layer reference picture set	475
G.8.2	NAL unit decoding process	475
G.8.3	Slice decoding processes	475
G.8.4	Decoding process for coding units coded in intra prediction mode	475
G.8.5	Decoding process for coding units coded in inter prediction mode	475
G.8.6	Scaling, transformation and array construction process prior to deblocking filter process	475
G.8.7	In-loop filter process	475
G.9	Parsing process	475
G.10	Specification of bitstream subsets	476
G.11	Profiles, tiers, and levels	476
G.11.1	Profiles	476
G.11.1.1	Multiview Main profile	476
G.11.2	Tiers and levels	478
G.11.2.1	General tier and level limits	478
G.11.2.2	Profile-specific tier and level limits for the Multiview Main profile	479
G.11.3	Decoder capabilities	480
G.12	Byte stream format	480
G.13	Hypothetical reference decoder	480
G.14	Supplemental enhancement information	480
G.14.1	General	480

G.14.2	SEI payload syntax.....	480
G.14.2.1	General SEI payload syntax.....	480
G.14.2.2	Annex D and Annex F SEI message syntax for multiview high efficiency video coding	480
G.14.2.3	3D reference displays information SEI message syntax	481
G.14.2.4	Depth representation information SEI message syntax.....	482
G.14.2.5	Multiview scene information SEI message syntax	482
G.14.2.6	Multiview acquisition information SEI message syntax.....	483
G.14.2.7	Multiview view position SEI message syntax	484
G.14.3	SEI payload semantics	484
G.14.3.1	General SEI payload semantics.....	484
G.14.3.2	Annex D and Annex F SEI message semantics for multiview high efficiency video coding	484
G.14.3.3	3D reference displays information SEI message semantics	484
G.14.3.4	Depth representation information SEI message semantics	487
G.14.3.5	Multiview scene information SEI message semantics	489
G.14.3.6	Multiview acquisition information SEI message semantics.....	490
G.14.3.7	Multiview view position SEI message semantics	493
G.15	Video usability information	493
Annex H	Scalable high efficiency video coding.....	494
H.1	Scope.....	494
H.2	Normative references	494
H.3	Definitions	494
H.4	Abbreviations.....	494
H.5	Conventions	494
H.6	Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships.....	494
H.7	Syntax and semantics	494
H.8	Decoding processes.....	494
H.8.1	General decoding process	494
H.8.1.1	General.....	494
H.8.1.2	Decoding process for a coded picture with nuh_layer_id greater than 0	494
H.8.1.3	Decoding process for inter-layer reference picture set.....	495
H.8.1.4	Derivation process for inter-layer reference pictures.....	495
H.8.2	NAL unit decoding process.....	507
H.8.3	Slice decoding processes.....	507
H.8.3.1	Decoding process for picture order count	507
H.8.3.2	Decoding process for reference picture set	507
H.8.3.3	Decoding process for generating unavailable reference pictures.....	507
H.8.3.4	Decoding process for reference picture lists construction.....	507
H.8.4	Decoding process for coding units coded in intra prediction mode	507
H.8.5	Decoding process for coding units coded in inter prediction mode	507
H.8.6	Scaling, transformation and array construction process prior to deblocking filter process.....	508
H.8.7	In-loop filter process	508
H.9	Parsing process	508
H.10	Specification of bitstream subsets.....	508
H.11	Profiles, tiers, and levels	508
H.11.1	Profiles	508
H.11.1.1	Scalable Main and Scalable Main 10 profiles.....	508
H.11.2	Tiers and levels	510
H.11.2.1	General tier and level limits	510
H.11.2.2	Profile specific tier and level limits for the Scalable Main and Scalable Main 10 profiles	512
H.11.3	Decoder capabilities	513
H.12	Byte stream format.....	513
H.13	Hypothetical reference decoder	513
H.14	Supplemental enhancement information.....	513
H.15	Video usability information	513
Bibliography		514

LIST OF FIGURES

Figure 6-1 – Nominal vertical and horizontal locations of 4:2:0 luma and chroma samples in a picture	18
--	----

Figure 6-2 – Nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture	18
Figure 6-3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture	19
Figure 6-4 – A picture with 11 by 9 luma coding tree blocks that is partitioned into two slices, the first of which is partitioned into three slice segments (informative)	20
Figure 6-5 – A picture with 11 by 9 luma coding tree blocks that is partitioned into two tiles and one slice (left) or is partitioned into two tiles and three slices (right) (informative)	20
Figure 7-1 – Structure of an access unit not containing any NAL units with nal_unit_type equal to FD_NUT, SUFFIX_SEI_NUT, VPS_NUT, SPS_NUT, PPS_NUT, RSV_VCL_N10, RSV_VCL_R11, RSV_VCL_N12, RSV_VCL_R13, RSV_VCL_N14, RSV_VCL_R15, RSV_IRAP_VCL22, or RSV_IRAP_VCL23, or in the range of RSV_VCL24..RSV_VCL31, RSV_NVCL41..RSV_NVCL47, or UNSPEC48..UNSPEC63	66
Figure 8-1 – Intra prediction mode directions (informative)	114
Figure 8-2 – Intra prediction angle definition (informative)	122
Figure 8-3 – Spatial motion vector neighbours (informative)	137
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	145
Figure 8-5 – Integer samples (shaded blocks with upper-case letters) and fractional sample positions (un-shaded blocks with lower-case letters) for eighth sample chroma interpolation	147
Figure 9-1 – Illustration of CABAC parsing process for a syntax element synEl (informative)	179
Figure 9-2 – Spatial neighbour T that is used to invoke the coding tree block availability derivation process relative to the current coding tree block (informative)	180
Figure 9-3 – Illustration of CABAC initialization process (informative)	181
Figure 9-4 – Illustration of CABAC storage process (informative)	191
Figure 9-5 – Overview of the arithmetic decoding process for a single bin (informative)	206
Figure 9-6 – Flowchart for decoding a decision	208
Figure 9-7 – Flowchart of renormalization	210
Figure 9-8 – Flowchart of bypass decoding process	211
Figure 9-9 – Flowchart of decoding a decision before termination	212
Figure 9-10 – Flowchart for encoding a decision	214
Figure 9-11 – Flowchart of renormalization in the encoder	215
Figure 9-12 – Flowchart of PutBit(B)	215
Figure 9-13 – Flowchart of encoding bypass	216
Figure 9-14 – Flowchart of encoding a decision before termination	217
Figure 9-15 – Flowchart of flushing at termination	217
Figure C.1 – Structure of byte streams and NAL unit streams for HRD conformance checks	238
Figure C.2 – HRD buffer model	241
Figure D.1 – Nominal vertical and horizontal sampling locations of 4:2:0 samples in top and bottom fields	280
Figure D.2 – Nominal vertical and horizontal sampling locations of 4:2:2 samples in top and bottom fields	281
Figure D.3 – Nominal vertical and horizontal sampling locations of 4:4:4 samples in top and bottom fields	281
Figure D.4 – Rearrangement and upconversion of side-by-side packing arrangement with frame_packing_arrangement_type equal to 3, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (4, 8) for both constituent frames	303
Figure D.5 – Rearrangement and upconversion of side-by-side packing arrangement with frame_packing_arrangement_type equal to 3, quincunx_sampling_flag equal to 0, (x, y) equal to (12, 8) for constituent frame 0, and (x, y) equal to (0, 0) or (4, 8) for constituent frame 1	304

Figure D.6 – Rearrangement and upconversion of top-bottom packing arrangement with frame_packing_arrangement_type equal to 4, quincunx_sampling_flag equal to 0, and (x, y) equal to (0, 0) or (8, 4) for both constituent frames.....	304
Figure D.7 – Rearrangement and upconversion of top-bottom packing arrangement with frame_packing_arrangement_type equal to 4, quincunx_sampling_flag equal to 0, (x, y) equal to (8, 12) for constituent frame 0, and (x, y) equal to (0, 0) or (8, 4) for constituent frame 1.....	305
Figure D.8 – Rearrangement and upconversion of side-by-side packing arrangement with quincunx sampling (frame_packing_arrangement_type equal to 3 with quincunx_sampling_flag equal to 1).....	305
Figure D.9 – Rearrangement of a temporal interleaving frame arrangement (frame_packing_arrangement_type equal to 5).....	306
Figure D.10 – Rearrangement of a segmented rectangular frame packing arrangement.....	320
Figure D.11 – A knee function with num_knee_points_minus1 equal to 2.....	333
Figure E.1 – Location of chroma samples for top and bottom fields for chroma_format_idc equal to 1 (4:2:0 chroma format) as a function of chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field.....	351
Figure F.1 – Bitstream-partition-specific HRD buffer model.....	435

LIST OF TABLES

Table 5-1 – Operation precedence from highest (at top of table) to lowest (at bottom of table).....	14
Table 6-1 – SubWidthC, and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag...	17
Table 7-1 – NAL unit type codes and NAL unit type classes.....	60
Table 7-2 – Interpretation of pic_type.....	79
Table 7-3 – Specification of sizeId.....	83
Table 7-4 – Specification of matrixId according to sizeId, prediction mode and colour component.....	83
Table 7-5 – Specification of default values of ScalingList[0][matrixId][i] with i = 0..15.....	84
Table 7-6 – Specification of default values of ScalingList[1..3][matrixId][i] with i = 0..63.....	84
Table 7-7 – Name association to slice_type.....	86
Table 7-8 – Specification of the SAO type.....	95
Table 7-9 – Specification of the SAO edge offset class.....	96
Table 7-10 – Name association to prediction mode and partitioning type.....	97
Table 7-11 – Name association to inter prediction mode.....	98
Table 8-1 – Specification of intra prediction mode and associated names.....	113
Table 8-2 – Specification of modeIdx.....	116
Table 8-3 – Specification of intraPredModeC when ChromaArrayType is equal to 2.....	116
Table 8-4 – Specification of intraHorVerDistThres[nTbS] for various transform block sizes.....	120
Table 8-5 – Specification of intraPredAngle.....	122
Table 8-6 – Specification of invAngle.....	122
Table 8-7 – Specification of l0CandIdx and l1CandIdx.....	135
Table 8-8 – Assignment of the luma prediction sample predSampleLX _L	146
Table 8-9 – Assignment of the chroma prediction sample predSampleLX _C for (X, Y) being replaced by (1, b), (2, c), (3, d), (4, e), (5, f), (6, g), and (7, h), respectively.....	148
Table 8-10 – Specification of Qp _C as a function of qPi for ChromaArrayType equal to 1.....	155
Table 8-11 – Name of association to edgeType.....	161

Table 8-12 – Derivation of threshold variables β' and t_c' from input Q	169
Table 8-13 – Specification of hPos and vPos according to the sample adaptive offset class	175
Table 9-1 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)	176
Table 9-2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative).....	177
Table 9-3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v).....	177
Table 9-4 – Association of ctxIdx and syntax elements for each initializationType in the initialization process	183
Table 9-5 – Values of initValue for ctxIdx of sao_merge_left_flag and sao_merge_up_flag	184
Table 9-6 – Values of initValue for ctxIdx of sao_type_idx_luma and sao_type_idx_chroma.....	184
Table 9-7 – Values of initValue for ctxIdx of split_cu_flag	184
Table 9-8 – Values of initValue for ctxIdx of cu_transquant_bypass_flag	184
Table 9-9 – Values of initValue for ctxIdx of cu_skip_flag	184
Table 9-10 – Values of initValue for ctxIdx of pred_mode_flag.....	185
Table 9-11 – Values of initValue for ctxIdx of part_mode.....	185
Table 9-12 – Values of initValue for ctxIdx of prev_intra_luma_pred_flag	185
Table 9-13 – Values of initValue for ctxIdx of intra_chroma_pred_mode.....	185
Table 9-14 – Values of initValue for ctxIdx of rqt_root_cbf.....	185
Table 9-15 – Value of initValue for ctxIdx of merge_flag	185
Table 9-16 – Values of initValue for ctxIdx of merge_idx.....	186
Table 9-17 – Values of initValue for ctxIdx of inter_pred_idc	186
Table 9-18 – Values of initValue for ctxIdx of ref_idx_l0 and ref_idx_l1	186
Table 9-19 – Values of initValue for ctxIdx of mvp_l0_flag and mvp_l1_flag	186
Table 9-20 – Values of initValue for ctxIdx of split_transform_flag	186
Table 9-21 – Values of initValue for ctxIdx of cbf_luma.....	186
Table 9-22 – Values of initValue for ctxIdx of cbf_cb and cbf_cr	187
Table 9-23 – Values of initValue for ctxIdx of abs_mvd_greater0_flag and abs_mvd_greater1_flag.....	187
Table 9-24 – Values of initValue for ctxIdx of cu_qp_delta_abs.....	187
Table 9-25 – Values of initValue for ctxIdx of transform_skip_flag	187
Table 9-26 – Values of initValue for ctxIdx of last_sig_coeff_x_prefix.....	187
Table 9-27 – Values of initValue for ctxIdx of last_sig_coeff_y_prefix.....	188
Table 9-28 – Values of initValue for ctxIdx of coded_sub_block_flag.....	188
Table 9-29 – Values of initValue for ctxIdx of sig_coeff_flag.....	188
Table 9-30 – Values of initValue for ctxIdx of coeff_abs_level_greater1_flag	189
Table 9-31 – Values of initValue for ctxIdx of coeff_abs_level_greater2_flag	189
Table 9-32 – Values of initValue for ctxIdx of explicit_rdpem_flag	189
Table 9-33 – Values of initValue for ctxIdx of explicit_rdpem_dir_flag.....	189
Table 9-34 – Values of initValue for ctxIdx of cu_chroma_qp_offset_flag.....	189
Table 9-35 – Values of initValue for ctxIdx of cu_chroma_qp_offset_idx.....	190
Table 9-36 – Values of initValue for ctxIdx of log2_res_scale_abs_plus1	190
Table 9-37 – Values of initValue for ctxIdx of res_scale_sign_flag	190

Table 9-38 – Syntax elements and associated binarizations	192
Table 9-39 – Bin string of the unary binarization (informative).....	195
Table 9-40 – Binarization for part_mode.....	197
Table 9-41 – Binarization for intra_chroma_pred_mode.....	197
Table 9-42 – Binarization for inter_pred_idc	198
Table 9-43 – Assignment of ctxInc to syntax elements with context coded bins	200
Table 9-44 – Specification of ctxInc using left and above syntax elements	202
Table 9-45 – Specification of ctxIdxMap[i]	204
Table 9-46 – Specification of rangeTabLps depending on the values of pStateIdx and qRangeIdx	209
Table 9-47 – State transition table	210
Table A.1 – Allowed values for syntax elements in the format range extensions profiles	223
Table A.2 – Bitstream indications for conformance to format range extensions profiles.....	225
Table A.3 – Bitstream indications for conformance to format range extensions high throughput profiles	227
Table A.4 – General tier and level limits	229
Table A.5 – Tier and level limits for the video profiles.....	231
Table A.6 – Specification of CpbVclFactor, CpbNalFactor, FormatCapabilityFactor, and MinCrScaleFactor	232
Table A.7 – Maximum picture rates (pictures per second) at level 1 to 4.1 for some example picture sizes when MinCbSizeY is equal to 64.....	233
Table A.8 – Maximum picture rates (pictures per second) at level 5 to 6.2 for some example picture sizes when MinCbSizeY is equal to 64.....	234
Table D.1 – Persistence scope of SEI messages (informative).....	273
Table D.2 – Interpretation of pic_struct.....	280
Table D.3 – scene_transition_type values	287
Table D.4 – film_grain_model_id values	289
Table D.5 – blending_mode_id values	291
Table D.6 – filter_hint_type values	295
Table D.7 – Interpretation of camera_iso_speed_idc and exposure_index_idc.....	298
Table D.8 – Definition of frame_packing_arrangement_type	300
Table D.9 – Definition of content_interpretation_type.....	301
Table D.10 – Interpretation of hash_type	308
Table D.11 – Definition of counting_type[i] values	316
Table D.12 – Definition of segmented_rect_content_interpretation_type.....	319
Table D.13 – ver_chroma_filter_idc values.....	324
Table D.14 – hor_chroma_filter_idc values	324
Table D.15 – Chroma sampling format indicated by target_format_idc	325
Table D.16 – Constraints on the value of num_vertical_filters	325
Table D.17 – Constraints on the value of num_horizontal_filters	326
Table D.18 – Values of verFilterCoeff and verTapLength when ver_chroma_filter_idc is equal to 2.....	327
Table D.19 – Values of horFilterCoeff and horTapLength when hor_chroma_filter_idc is equal to 2.....	327
Table D.20 – Usage of chroma filter in the vertical direction	330

Table D.21 – Usage of chroma filter in the horizontal direction	332
Table E.1 – Interpretation of sample aspect ratio indicator	342
Table E.2 – Meaning of video_format.....	343
Table E.3 – Colour primaries.....	344
Table E.4 – Transfer characteristics	346
Table E.5 – Matrix coefficients	350
Table E.6 – Divisor for computation of DpbOutputElementalInterval[n]	357
Table F.1 – Mapping of ScalabiltyId to scalability dimensions.....	386
Table F.2 – Mapping of AuxId to the type of auxiliary pictures	387
Table F.3 – Specification of CompatibleProfileList	432
Table F.4 – Persistence scope of SEI messages (informative).....	454
Table G.1 – Persistence scope of SEI messages (informative).....	484
Table G.2 – Association between camera parameter variables and syntax elements	486
Table G.3 – Definition of depth_representation_type	488
Table G.4 – Association between depth parameter variables and syntax elements	488
Table G.5 – Association between camera parameter variables and syntax elements.	493
Table H.1 – 16-phase luma resampling filter.....	500
Table H.2 – 16-phase chroma resampling filter.....	501

0 Introduction

0.1 General

This clause and its subclauses do not form an integral part of this Recommendation | International Standard.

0.2 Prologue

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 Collaborative Team on Video Coding (JCT-VC) in 2010 for development of a new Recommendation | International Standard. This Recommendation | International Standard was developed in the JCT-VC.

0.3 Purpose

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 communications. It is also designed to enable the use of the coded video representation in a flexible manner for a wide variety of network environments as well as to enable the use of multi-core parallel encoding and decoding devices. 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. Supports for higher bit depths and enhanced chroma formats, including the use of full-resolution chroma are provided. Support for scalability enables video transmission on networks with varying transmission conditions and other scenarios involving multiple bit rate services. Support for multiview enables representation of video content with multiple camera views and optional auxiliary information.

0.4 Applications

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

- Broadcast (cable TV on optical networks / copper, satellite, terrestrial, etc.)
- Camcorders
- Content production and distribution
- Digital cinema
- Home cinema
- Internet streaming, download and play
- Medical imaging
- Mobile streaming, broadcast and communications
- Real-time conversational services (videoconferencing, videophone, telepresence, etc.)
- Remote video surveillance
- Storage media (optical disks, digital video tape recorder, etc.)
- Wireless display

0.5 Publication and versions of this Specification

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

the basis text has been drafted to become both an ITU-T Recommendation and an ISO/IEC International Standard, the term "Specification" (with capitalization to indicate that it refers to the whole of the text) is used herein when the text refers to itself.

This is the second version of this Specification. Additional versions are anticipated.

Rec. ITU-T H.265 | ISO/IEC 23008-2 version 1 refers to the first approved version of this Recommendation | International Standard.

Rec. ITU-T H.265 | ISO/IEC 23008-2 version 2 refers to the integrated text containing format range extensions, scalability extensions, multiview extensions, additional supplement enhancement information, and corrections to various minor defects in the prior content of the specification.

0.6 Profiles, tiers, and levels

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", "tiers", 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 in 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, "tiers" and "levels" are specified within each profile. A level of a tier 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). A level specified for a lower tier is more constrained than a level specified for a higher tier.

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.

0.7 Overview of the design characteristics

The coded representation specified in the syntax is designed to enable a high compression capability for a desired image or video quality. 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 may then be further compressed using a transform to remove spatial correlation inside the transform block before it is quantized, producing a possibly 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 may also be further compressed using a variety of prediction mechanisms, and, after prediction, are combined with the quantized transform coefficient information and encoded using arithmetic coding.

0.8 How to read this Specification

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 clauses 7.1–7.3 for syntactical order and see clause 7.4 for semantics; e.g., 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). Clause 10 (Sub-bitstream extraction process) specifies the sub-bitstream extraction 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 profiles each being tailored to certain application domains, and defines the so-called tiers and 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, bitstream conformance, decoder conformance, and the use of the hypothetical reference decoder to check bitstream and decoder conformance. Annex D specifies syntax and semantics for supplemental enhancement information message payloads. Annex E specifies syntax and semantics of the video usability information parameters of the sequence parameter set. Annex F specifies general multi-layer support for bitstreams and decoders. Annex G contains support for multiview coding. Annex H contains support for scalability.

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

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 2: High efficiency video coding

1 Scope

This Recommendation | International Standard specifies high efficiency video coding.

2 Normative references

2.1 General

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.

2.2 Identical Recommendations | International Standards

- None

2.3 Paired Recommendations | International Standards equivalent in technical content

- None

2.4 Additional references

- Recommendation ITU-T T.35 (in force), *Procedure for the allocation of ITU-T defined codes for non-standard facilities*.
- ISO/IEC 11578: in force, *Information technology — Open Systems Interconnection — Remote Procedure Call (RPC)*.
- ISO 11664-1: in force, *Colorimetry — Part 1: CIE standard colorimetric observers*.
- ISO 12232: in force, *Photography — Digital still cameras — Determination of exposure index, ISO speed ratings, standard output sensitivity, and recommended exposure index*.
- IETF RFC 1321 (in force), *The MD5 Message-Digest Algorithm*.
- ISO/IEC 10646: in force, *Information technology — Universal Coded Character Set (UCS)*.
- IETF RFC 5646 (in force), *Tags for Identifying Languages*.

3 Definitions

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

- 3.1 access unit:** A set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order*, and contain exactly one *coded picture* with *nuh_layer_id* equal to 0.

NOTE 1 – In addition to containing the VCL NAL units of the coded picture with *nuh_layer_id* equal to 0, an access unit may also contain non-VCL NAL units. The decoding of an access unit with the decoding process specified in clause 8 always results in a decoded picture with *nuh_layer_id* equal to 0.

NOTE 2 – An access unit is defined differently in Annex F and does not need to contain a coded picture with *nuh_layer_id* equal to 0.

- 3.2 AC transform coefficient:** Any *transform coefficient* for which the *frequency index* in at least one of the two dimensions is non-zero.

- 3.3 associated non-VCL NAL unit:** A *non-VCL NAL unit* (when present) for a *VCL NAL unit* where the *VCL NAL unit* is the *associated VCL NAL unit* of the *non-VCL NAL unit*.
- 3.4 associated IRAP picture:** The previous *IRAP picture* in *decoding order* (when present).
- 3.5 associated VCL NAL unit:** The preceding *VCL NAL unit* in *decoding order* for a *non-VCL NAL unit* with *nal_unit_type* equal to EOS_NUT, EOB_NUT, FD_NUT, or SUFFIX_SEI_NUT, or in the ranges of RSV_NVCL45..RSV_NVCL47 or UNSPEC56..UNSPEC63; or otherwise the next *VCL NAL unit* in *decoding order*.
- 3.6 base layer:** A *layer* in which all *NAL units* have *nuh_layer_id* equal to 0.
- 3.7 bin:** One bit of a *bin string*.
- 3.8 binarization:** A set of *bin strings* for all possible values of a *syntax element*.
- 3.9 binarization process:** A unique mapping process of all possible values of a *syntax element* onto a set of *bin strings*.
- 3.10 bin string:** An intermediate binary representation of values of *syntax elements* from the *binarization* of the *syntax element*.
- 3.11 bi-predictive (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*.
- 3.12 bitstream:** A sequence of bits, in the form of a *NAL unit stream* or a *byte stream*, that forms the representation of *coded pictures* and associated data forming one or more *CVSs*.
- 3.13 block:** An MxN (M-column by N-row) array of samples, or an MxN array of *transform coefficients*.
- 3.14 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*.
- 3.15 broken link access (BLA) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is a *BLA picture*.
- 3.16 broken link access (BLA) picture:** An *IRAP picture* for which each *VCL NAL unit* has *nal_unit_type* equal to BLA_W_LP, BLA_W_RADL, or BLA_N_LP.
- NOTE – A BLA picture contains only I slices, and may be the first picture in the bitstream in decoding order, or may appear later in the bitstream. Each BLA picture begins a new CVS, and has the same effect on the decoding process as an IDR picture. However, a BLA picture contains syntax elements that specify a non-empty RPS. When a BLA picture for which each VCL NAL unit has *nal_unit_type* equal to BLA_W_LP, it may have associated RASL pictures, which are not output by the decoder and may not be decodable, as they may contain references to pictures that are not present in the bitstream. When a BLA picture for which each VCL NAL unit has *nal_unit_type* equal to BLA_W_LP, it may also have associated RADL pictures, which are specified to be decoded. When a BLA picture for which each VCL NAL unit has *nal_unit_type* equal to BLA_W_RADL, it does not have associated RASL pictures but may have associated RADL pictures. When a BLA picture for which each VCL NAL unit has *nal_unit_type* equal to BLA_N_LP, it does not have any associated leading pictures.
- 3.17 buffering period:** The set of *access units* starting with an *access unit* that contains a buffering period SEI message and containing all subsequent *access units* in *decoding order* up to but not including the next *access unit* (when present) that contains a buffering period SEI message.
- 3.18 byte:** A sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively.
- 3.19 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*, and 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.
- 3.20 byte stream:** An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units* as specified in Annex B.
- 3.21 can:** A term used to refer to behaviour that is allowed, but not necessarily required.
- 3.22 chroma:** An adjective, represented by the symbols Cb and Cr, specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours.
- 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.
- 3.23 clean random access (CRA) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is a *CRA picture*.

- 3.24 clean random access (CRA) picture:** An *IRAP picture* for which each *VCL NAL unit* has *nal_unit_type* equal to *CRA_NUT*.
- NOTE – A CRA picture contains only I slices, and may be the first picture in the bitstream in decoding order, or may appear later in the bitstream. A CRA picture may have associated RADL or RASL pictures. When a CRA picture has *NoRaslOutputFlag* equal to 1, the associated RASL pictures are not output by the decoder, because they may not be decodable, as they may contain references to pictures that are not present in the bitstream.
- 3.25 coded picture:** A *coded representation* of a *picture* containing all *coding tree units* of the *picture*.
- 3.26 coded picture buffer (CPB):** A first-in first-out buffer containing *decoding units* in *decoding order* specified in the *hypothetical reference decoder* in Annex C.
- 3.27 coded representation:** A data element as represented in its coded form.
- 3.28 coded slice segment NAL unit:** A *NAL unit* that has *nal_unit_type* in the range of *TRAIL_N* to *RASL_R*, inclusive, or in the range of *BLA_W_LP* to *RSV_IRAP_VCL23*, inclusive, which indicates that the *NAL unit* contains a coded *slice segment*.
- 3.29 coded layer-wise video sequence (CLVS):** A sequence of *pictures* and the *associated non-VCL NAL units* of the *base layer* of a *CVS*.
- 3.30 coded video sequence (CVS):** A sequence of *access units* that consists, in *decoding order*, of an *IRAP access unit* with *NoRaslOutputFlag* equal to 1, followed by zero or more *access units* that are not *IRAP access units* with *NoRaslOutputFlag* equal to 1, including all subsequent *access units* up to but not including any subsequent *access unit* that is an *IRAP access unit* with *NoRaslOutputFlag* equal to 1.
- NOTE – An *IRAP access unit* may be an *IDR access unit*, a *BLA access unit*, or a *CRA access unit*. The value of *NoRaslOutputFlag* is equal to 1 for each *IDR access unit*, each *BLA access unit*, and each *CRA access unit* that is the first *access unit* in the bitstream in decoding order, is the first *access unit* that follows an end of sequence *NAL unit* in decoding order, or has *HandleCraAsBlaFlag* equal to 1.
- 3.31 coded video sequence group (CVSG):** One or more consecutive *CVSs* in *decoding order* that collectively consist of an *IRAP access unit* that activates a VPS RBSP *firstVpsRbsp* that was not already active followed by all subsequent *access units*, in *decoding order*, for which *firstVpsRbsp* is the active VPS RBSP up to the end of the *bitstream* or up to but excluding the *access unit* that activates a different VPS RBSP than *firstVpsRbsp*, whichever is earlier in *decoding order*.
- 3.32 coding block:** An *NxN block* of samples for some value of *N* such that the division of a *coding tree block* into *coding blocks* is a *partitioning*.
- 3.33 coding tree block:** An *NxN block* of samples for some value of *N* such that the division of a *component* into *coding tree blocks* is a *partitioning*.
- 3.34 coding tree unit:** A *coding tree block* of *luma* samples, two corresponding *coding tree blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *coding tree block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to code the samples.
- 3.35 coding unit:** A *coding block* of *luma* samples, two corresponding *coding blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *coding block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to code the samples.
- 3.36 component:** An array or single sample from one of the three arrays (*luma* and two *chroma*) that compose a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* in monochrome format.
- 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 cropped decoded picture:** The result of cropping a *decoded picture* based on the conformance cropping window specified in the *SPS* that is referred to by the corresponding *coded picture*.
- 3.39 decoded picture:** A *decoded picture* is derived by decoding a *coded picture*.
- 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 decoder under test (DUT):** A *decoder* that is tested for conformance to this Specification 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 or order of the output of the two *decoders*.

- 3.43 decoding order:** The order in which *syntax elements* are processed by the *decoding process*.
- 3.44 decoding process:** The process specified in this Specification that reads a *bitstream* and derives *decoded pictures* from it.
- 3.45 decoding unit:** An *access unit* if SubPicHrdFlag is equal to 0 or a subset of an *access unit* otherwise, consisting of one or more *VCL NAL units* in an *access unit* and the *associated non-VCL NAL units*.
- 3.46 dependent slice segment:** A *slice segment* for which the values of some *syntax elements* of the *slice segment header* are inferred from the values for the preceding *independent slice segment* in *decoding order*.
- 3.47 display process:** A process not specified in this Specification having, as its input, the *cropped decoded pictures* that are the output of the *decoding process*.
- 3.48 elementary stream:** A sequence of one or more *bitstreams*.
NOTE – An elementary stream that consists of two or more bitstreams would typically have been formed by splicing together two or more bitstreams (or parts thereof).
- 3.49 emulation prevention byte:** A *byte* equal to 0x03 that is present within a *NAL unit* when the *syntax elements* of the *bitstream* form certain patterns of *byte* values in a manner that ensures that no sequence of consecutive *byte-aligned bytes* in the *NAL unit* can contain a *start code prefix*.
- 3.50 encoder:** An embodiment of an *encoding process*.
- 3.51 encoding process:** A process not specified in this Specification that produces a *bitstream* conforming to this Specification.
- 3.52 field:** An assembly of alternative rows of samples of a *frame*.
- 3.53 filler data NAL units:** *NAL units* with *nal_unit_type* equal to FD_NUT.
- 3.54 flag:** A variable or single-bit *syntax element* that can take one of the two possible values: 0 and 1.
- 3.55 frame:** The composition of a top *field* and a bottom *field*, where sample rows 0, 2, 4, ... originate from the top *field* and sample rows 1, 3, 5, ... originate from the bottom *field*.
- 3.56 frequency index:** A one-dimensional or two-dimensional index associated with a *transform coefficient* prior to an *inverse transform* part of the *decoding process*.
- 3.57 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.
- 3.58 hypothetical stream scheduler (HSS):** A hypothetical delivery mechanism used for checking the conformance of a *bitstream* or a *decoder* with regards to the timing and data flow of the input of a *bitstream* into the *hypothetical reference decoder*.
- 3.59 independent slice segment:** A *slice segment* for which the values of the *syntax elements* of the *slice segment header* are not inferred from the values for a preceding *slice segment*.
- 3.60 informative:** A term used to refer to content provided in this Specification that does not establish any mandatory requirements for conformance to this Specification and thus is not considered an integral part of this Specification.
- 3.61 instantaneous decoding refresh (IDR) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is an *IDR picture*.
- 3.62 instantaneous decoding refresh (IDR) picture:** An *IRAP picture* for which each *VCL NAL unit* has *nal_unit_type* equal to IDR_W_RADL or IDR_N_LP.
NOTE – An IDR picture contains only I slices, and may be the first picture in the bitstream in decoding order, or may appear later in the bitstream. Each IDR picture is the first picture of a CVS in decoding order. When an IDR picture for which each VCL NAL unit has *nal_unit_type* equal to IDR_W_RADL, it may have associated RADL pictures. When an IDR picture for which each VCL NAL unit has *nal_unit_type* equal to IDR_N_LP, it does not have any associated leading pictures. An IDR picture does not have associated RASL pictures.
- 3.63 inter coding:** Coding of a *coding block*, *slice*, or *picture* that uses *inter prediction*.
- 3.64 inter prediction:** A *prediction* derived in a manner that is dependent on data elements (e.g., sample values or motion vectors) of *pictures* other than the current *picture*.
- 3.65 intra coding:** Coding of a *coding block*, *slice*, or *picture* that uses *intra prediction*.
- 3.66 intra prediction:** A *prediction* derived from only data elements (e.g., sample values) of the same decoded *slice*.

- 3.67 intra random access point (IRAP) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is an *IRAP picture*.
- 3.68 intra random access point (IRAP) picture:** A *coded picture* for which each *VCL NAL unit* has *nal_unit_type* in the range of BLA_W_LP to RSV_IRAP_VCL23, inclusive.
- NOTE – An IRAP picture contains only I slices, and may be a BLA picture, a CRA picture or an IDR picture. The first picture in the bitstream in decoding order must be an IRAP picture. Provided the necessary parameter sets are available when they need to be activated, the IRAP picture and all subsequent non-RASL pictures in decoding order can be correctly decoded without performing the decoding process of any pictures that precede the IRAP picture in decoding order. There may be pictures in a bitstream that contain only I slices that are not IRAP pictures.
- 3.69 intra (I) slice:** A *slice* that is decoded using *intra prediction* only.
- 3.70 inverse transform:** A part of the *decoding process* by which a set of *transform coefficients* are converted into spatial-domain values.
- 3.71 layer:** A set of *VCL NAL units* that all have a particular value of *nuh_layer_id* and the *associated non-VCL NAL units*, or one of a set of syntactical structures having a hierarchical relationship.
- NOTE – Depending on the context, either the first layer concept or the second layer concept applies. The first layer concept is also referred to as a scalable layer, wherein a layer may be a spatial scalable layer, a quality scalable layer, a view, etc. A temporal true subset of a scalable layer is not referred to as a layer but referred to as a sub-layer or temporal sub-layer. The second layer concept is also referred to as a coding layer, wherein higher layers contain lower layers, and the coding layers are the CVS, picture, slice, slice segment, and coding tree unit layers.
- 3.72 layer identifier list:** A list of *nuh_layer_id* values that is associated with a *layer set* or an *operation point* and can be used as an input to the *sub-bitstream extraction process*.
- 3.73 layer set:** A set of *layers* represented within a *bitstream* created from another *bitstream* by operation of the *sub-bitstream extraction process* with the another *bitstream*, the target highest TemporalId equal to 6, and the target *layer identifier list* equal to the *layer identifier list* associated with the layer set as inputs.
- 3.74 leading picture:** A *picture* that precedes the *associated IRAP picture* in *output order*.
- 3.75 leaf:** A terminating node of a tree that is a root node of a tree of depth 0.
- 3.76 level:** A defined set of constraints on the values that may be taken by the *syntax elements* and variables of this Specification, or the value of a *transform coefficient* prior to *scaling*.
- NOTE – 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 the specified constraints, support a different level for each supported profile.
- 3.77 list 0 (list 1) motion vector:** A *motion vector* associated with a *reference index* pointing into *reference picture list 0 (list 1)*.
- 3.78 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)*.
- 3.79 long-term reference picture:** A *picture* that is marked as "used for long-term reference".
- 3.80 long-term reference picture set:** The two RPS lists that may contain long-term reference pictures.
- 3.81 luma:** An adjective, represented by the symbol or subscript Y or L, specifying that a sample array or single sample is representing the monochrome signal related to the primary colours.
- 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.
- 3.82 may:** A term that is used to refer to behaviour that is allowed, but not necessarily required.
- NOTE – 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.
- 3.83 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*.
- 3.84 must:** A term that is used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Specification (used exclusively in an *informative* context).
- 3.85 nested SEI message:** An SEI message that is contained in a scalable nesting SEI message.

- 3.86 network abstraction layer (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*.
- 3.87 network abstraction layer (NAL) unit stream:** A sequence of *NAL units*.
- 3.88 non-nested SEI message:** An SEI message that is not contained in a scalable nesting SEI message.
- 3.89 non-reference picture:** A *picture* that is marked as "unused for reference".
 NOTE – A non-reference picture contains samples that cannot be used for inter prediction in the decoding process of subsequent pictures in decoding order. In other words, once a picture is marked as "unused for reference", it can never be marked back as "used for reference".
- 3.90 non-VCL NAL unit:** A *NAL unit* that is not a *VCL NAL unit*.
- 3.91 note:** A term that is used to prefix *informative* remarks (used exclusively in an *informative* context).
- 3.92 operation point:** A *bitstream* created from another *bitstream* by operation of the *sub-bitstream extraction process* with the another *bitstream*, a target highest TemporalId, and a target *layer identifier list* as inputs.
 NOTE – If the target highest TemporalId of an operation point is equal to the greatest value of TemporalId in the layer set associated with the target layer identification list, the operation point is identical to the layer set. Otherwise it is a subset of the layer set.
- 3.93 output order:** The order in which the *decoded pictures* are output from the *decoded picture buffer* (for the *decoded pictures* that are to be output from the *decoded picture buffer*).
- 3.94 output time:** A time when a *decoded picture* is to be output as specified by the *HRD* according to the output timing *DPB* operation.
- 3.95 parameter:** A *syntax element* of a *VPS*, *SPS* or *PPS*, or the second word of the defined term *quantization parameter*.
- 3.96 partitioning:** The division of a set into subsets such that each element of the set is in exactly one of the subsets.
- 3.97 picture:** 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.
 NOTE – A picture may be either a frame or a field. However, in one CVS, either all pictures are frames or all pictures are fields.
- 3.98 picture parameter set (PPS):** A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by a *syntax element* found in each *slice segment header*.
- 3.99 picture order count (POC):** A variable that is associated with each *picture*, uniquely identifies the associated *picture* among all *pictures* in the *CVS*, and, when the associated *picture* is to be output from the *decoded picture buffer*, indicates the position of the associated *picture* in *output order* relative to the *output order* positions of the other *pictures* in the same *CVS* that are to be output from the *decoded picture buffer*.
- 3.100 picture unit:** A set of *NAL units* that contain all *VCL NAL units* of a *coded picture* and their associated *non-VCL NAL units*.
- 3.101 prediction:** An embodiment of the *prediction process*.
- 3.102 prediction block:** A rectangular MxN *block* of samples on which the same *prediction* is applied.
- 3.103 prediction process:** The use of a *predictor* to provide an estimate of the data element (e.g., sample value or motion vector) currently being decoded.
- 3.104 prediction unit:** A *prediction block* of *luma* samples, two corresponding *prediction blocks* of *chroma* samples of a *picture* that has three sample arrays, or a *prediction block* of samples of a monochrome *picture* or a *picture* that is coded using three separate colour planes and *syntax structures* used to predict the *prediction block* samples.
- 3.105 predictive (P) slice:** A *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*.
- 3.106 predictor:** A combination of specified values or previously decoded data elements (e.g., sample value or motion vector) used in the *decoding process* of subsequent data elements.
- 3.107 prefix SEI message:** An SEI message that is contained in a *prefix SEI NAL unit*.
- 3.108 prefix SEI NAL unit:** An *SEI NAL unit* that has *nal_unit_type* equal to PREFIX_SEI_NUT.

- 3.109 profile:** A specified subset of the syntax of this Specification.
- 3.110 pulse code modulation (PCM):** Coding of the samples of a *block* by directly representing the sample values without *prediction* or application of a transform.
- 3.111 quadtree:** A *tree* in which a parent node can be split into four child nodes, each of which may become parent node for another split into four child nodes.
- 3.112 quantization parameter:** A variable used by the *decoding process* for *scaling of transform coefficient levels*.
- 3.113 random access:** The act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream.
- 3.114 random access decodable leading (RADL) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is a *RADL picture*.
- 3.115 random access decodable leading (RADL) picture:** A *coded picture* for which each *VCL NAL unit* has *nal_unit_type* equal to RADL_R or RADL_N.
NOTE – All RADL pictures are leading pictures. RADL pictures are not used as reference pictures for the decoding process of trailing pictures of the same associated IRAP picture. When present, all RADL pictures precede, in decoding order, all trailing pictures of the same associated IRAP picture.
- 3.116 random access skipped leading (RASL) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is a *RASL picture*.
- 3.117 random access skipped leading (RASL) picture:** A *coded picture* for which each *VCL NAL unit* has *nal_unit_type* equal to RASL_R or RASL_N.
NOTE – All RASL pictures are leading pictures of an associated BLA or CRA picture. When the associated IRAP picture has *NoRaslOutputFlag* equal to 1, the RASL picture is not output and may not be correctly decodable, as the RASL picture may contain references to pictures that are not present in the bitstream. RASL pictures are not used as reference pictures for the decoding process of non-RASL pictures. When present, all RASL pictures precede, in decoding order, all trailing pictures of the same associated IRAP picture.
- 3.118 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.
- 3.119 raw byte sequence payload (RBSP):** A *syntax structure* containing an integer number of *bytes* that is encapsulated in a *NAL unit* and that is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and zero or more subsequent bits equal to 0.
- 3.120 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*, for which the location of the end 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*.
- 3.121 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*.
- 3.122 reference index:** An index into a *reference picture list*.
- 3.123 reference picture:** A *picture* that is a *short-term reference picture* or a *long-term reference picture*.
NOTE – 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* or *B slice*.
NOTE – For the decoding process of a *P* slice, there is one reference picture list – reference picture list 0. For the decoding process of a *B* slice, there are two reference picture lists – reference picture list 0 and reference picture list 1.
- 3.125 reference picture list 0:** The *reference picture list* used for *inter prediction* of a *P* or the first *reference picture list* used for *inter prediction* of a *B slice*.
- 3.126 reference picture list 1:** The second *reference picture list* used for *inter prediction* of a *B slice*.
- 3.127 reference picture set (RPS):** A set of *reference pictures* associated with a *picture*, consisting of all *reference pictures* that are prior to the associated *picture* in *decoding order*, that may be used for *inter prediction* of the associated *picture* or any *picture* following the associated *picture* in *decoding order*.
NOTE – The RPS of a picture consists of five RPS lists, three of which are to contain short-term reference pictures and the other two are to contain long-term reference pictures.

- 3.128 reserved:** A term that may be used to specify that some values of a particular *syntax element* are for future use by ITU-T | ISO/IEC and shall not be used in *bitstreams* conforming to this version of this Specification, but may be used in bitstreams conforming to future extensions of this Specification 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 sample aspect ratio:** 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 *picture*, which is specified for assisting the *display process* (not specified in this Specification) and expressed as $h:v$, where h is the horizontal width and v is the vertical height, in arbitrary units of spatial distance.
- 3.131 scaling:** The process of multiplying *transform coefficient levels* by a factor, resulting in *transform coefficients*.
- 3.132 sequence parameter set (SPS):** A *syntax structure* containing *syntax elements* that apply to zero or more entire *CVSs* as determined by the content of a *syntax element* found in the *PPS* referred to by a *syntax element* found in each *slice segment header*.
- 3.133 shall:** A term used to express mandatory requirements for conformance to this Specification.
- NOTE – 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 cropped decoded pictures to those output from the decoding process described in this Specification conforms to the decoding process requirements of this Specification.
- 3.134 short-term reference picture:** A *picture* that is marked as "used for short-term reference".
- 3.135 short-term reference picture set:** The three RPS lists that may contain short-term reference pictures.
- 3.136 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 Specification.
- 3.137 slice:** An integer number of *coding tree units* contained in one *independent slice segment* and all subsequent *dependent slice segments* (if any) that precede the next *independent slice segment* (if any) within the same *access unit*.
- 3.138 slice header:** The *slice segment header* of the *independent slice segment* that is a current *slice segment* or the most recent *independent slice segment* that precedes a current *dependent slice segment* in *decoding order*.
- 3.139 slice segment:** An integer number of *coding tree units* ordered consecutively in the *tile scan* and contained in a single *NAL unit*.
- 3.140 slice segment header:** A part of a coded *slice segment* containing the data elements pertaining to the first or all *coding tree units* represented in the *slice segment*.
- 3.141 source:** A term used to describe the video material or some of its attributes before encoding.
- 3.142 start code prefix:** A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*.
- NOTE – 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.
- 3.143 step-wise temporal sub-layer access (STSA) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is an *STSA picture*.
- 3.144 step-wise temporal sub-layer access (STSA) picture:** A *coded picture* for which each *VCL NAL unit* has *nal_unit_type* equal to STSA_R or STSA_N.
- NOTE – An STSA picture does not use pictures with the same TemporalId as the STSA picture for inter prediction reference. Pictures following an STSA picture in decoding order with the same TemporalId as the STSA picture do not use pictures prior to the STSA picture in decoding order with the same TemporalId as the STSA picture for inter prediction reference. An STSA picture enables up-switching, at the STSA picture, to the sub-layer containing the STSA picture, from the immediately lower sub-layer. STSA pictures must have TemporalId greater than 0.
- 3.145 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*, where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.
- 3.146 sub-bitstream extraction process:** A specified process by which *NAL units* in a *bitstream* that do not belong to a target set, determined by a target highest TemporalId and a target *layer identifier list*, are removed from

the *bitstream*, with the output sub-bitstream consisting of the NAL units in the *bitstream* that belong to the target set.

- 3.147 sub-layer:** A temporal scalable layer of a temporal scalable *bitstream*, consisting of *VCL NAL units* with a particular value of the TemporalId variable and the associated *non-VCL NAL units*.
- 3.148 sub-layer non-reference (SLNR) picture:** A *picture* that contains samples that cannot be used for *inter prediction* in the *decoding process* of subsequent *pictures* of the same *sub-layer* in *decoding order*.
NOTE – Samples of an SLNR picture may be used for inter prediction in the decoding process of subsequent pictures of higher sub-layers in decoding order.
- 3.149 sub-layer reference picture:** A *picture* that contains samples that may be used for *inter prediction* in the *decoding process* of subsequent *pictures* of the same *sub-layer* in *decoding order*.
NOTE – Samples of a sub-layer reference picture may also be used for inter prediction in the decoding process of subsequent pictures of higher sub-layers in decoding order.
- 3.150 sub-layer representation:** A subset of the *bitstream* consisting of *NAL units* of a particular *sub-layer* and the lower *sub-layers*.
- 3.151 suffix SEI message:** An SEI message that is contained in a *suffix SEI NAL unit*.
- 3.152 suffix SEI NAL unit:** An *SEI NAL unit* that has *nal_unit_type* equal to SUFFIX_SEI_NUT.
- 3.153 supplemental enhancement information (SEI) NAL unit:** A *NAL unit* that has *nal_unit_type* equal to PREFIX_SEI_NUT or SUFFIX_SEI_NUT.
- 3.154 syntax element:** An element of data represented in the *bitstream*.
- 3.155 syntax structure:** Zero or more *syntax elements* present together in the *bitstream* in a specified order.
- 3.156 temporal sub-layer:** Same as *sub-layer*.
- 3.157 temporal sub-layer access (TSA) access unit:** An *access unit* in which the *coded picture* with *nuh_layer_id* equal to 0 is a *TSA picture*.
- 3.158 temporal sub-layer access (TSA) picture:** A *coded picture* for which each *VCL NAL unit* has *nal_unit_type* equal to TSA_R or TSA_N.
NOTE – A TSA picture and pictures following the TSA picture in decoding order do not use pictures prior to the TSA picture in decoding order with TemporalId greater than or equal to that of the TSA picture for inter prediction reference. A TSA picture enables up-switching, at the TSA picture, to the sub-layer containing the TSA picture or any higher sub-layer, from the immediately lower sub-layer. TSA pictures must have TemporalId greater than 0.
- 3.159 tier:** A specified category of *level* constraints imposed on values of the *syntax elements* in the *bitstream*, where the *level* constraints are nested within a *tier* and a *decoder* conforming to a certain *tier* and *level* would be capable of decoding all *bitstreams* that conform to the same *tier* or the lower *tier* of that *level* or any *level* below it.
- 3.160 tile:** A rectangular region of *coding tree blocks* within a particular *tile column* and a particular *tile row* in a *picture*.
- 3.161 tile column:** A rectangular region of *coding tree blocks* having a height equal to the height of the *picture* and a width specified by *syntax elements* in the *picture parameter set*.
- 3.162 tile row:** A rectangular region of *coding tree blocks* having a height specified by *syntax elements* in the *picture parameter set* and a width equal to the width of the *picture*.
- 3.163 tile scan:** A specific sequential ordering of *coding tree blocks* partitioning a *picture* in which the *coding tree blocks* are ordered consecutively in *coding tree block raster scan* in a *tile* whereas *tiles* in a *picture* are ordered consecutively in a *raster scan* of the *tiles* of the *picture*.
- 3.164 trailing picture:** A non-IRAP *picture* that follows the *associated IRAP picture* in *output order*.
NOTE – Trailing pictures associated with an IRAP picture also follow the IRAP picture in decoding order. Pictures that follow the associated IRAP picture in output order and precede the associated IRAP picture in decoding order are not allowed.
- 3.165 transform block:** A rectangular MxN *block* of samples on which the same *transform* is applied.
- 3.166 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*.

- 3.167 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.
- 3.168 transform unit:** A *transform block* of *luma* samples of size 8x8, 16x16, or 32x32 or four *transform blocks* of *luma* samples of size 4x4, two corresponding *transform blocks* of *chroma* samples of a *picture* in 4:2:0 colour format; or a *transform block* of *luma* samples of size 8x8, 16x16, or 32x32, and four corresponding *transform blocks* of *chroma* samples, or four *transform blocks* of *luma* samples of size 4x4, and four corresponding *transform blocks* of *chroma* samples of a *picture* in 4:2:2 colour format; or a *transform block* of *luma* samples of size 4x4, 8x8, 16x16, or 32x32, and two corresponding *transform blocks* of *chroma* samples of a *picture* in 4:4:4 colour format that is not coded using three separate colour planes and *syntax structures* used to transform the *transform block* samples; or a *transform block* of *luma* samples of size 8x8, 16x16, or 32x32 or four *transform blocks* of *luma* samples of size 4x4 of a monochrome *picture* or a *picture* in 4:4:4 colour format that is coded using three separate colour planes; and the associated *syntax structures* used to transform the *transform block* samples.
- 3.169 tree:** A tree is a finite set of nodes with a unique root node.
- 3.170 universal unique identifier (UUID):** An identifier that is unique with respect to the space of all universal unique identifiers.
- 3.171 unspecified:** A term that may be used to specify some values of a particular *syntax element* to indicate that the values have no specified meaning in this Specification and will not have a specified meaning in the future as an integral part of future versions of this Specification.
- 3.172 video coding layer (VCL) NAL unit:** A collective term for *coded slice segment NAL units* and the subset of *NAL units* that have *reserved* values of *nal_unit_type* that are classified as VCL NAL units in this Specification.
- 3.173 video parameter set (VPS):** A *syntax structure* containing *syntax elements* that apply to zero or more entire *CVSs* as determined by the content of a *syntax element* found in the *SPS* referred to by a *syntax element* found in the *PPS* referred to by a *syntax element* found in each *slice segment header*.
- 3.174 z-scan order:** A specified sequential ordering of *blocks partitioning a picture*, where the order is identical to *coding tree block raster scan* of the *picture* when the *blocks* are of the same size as *coding tree blocks*, and, when the *blocks* are of a smaller size than *coding tree blocks*, i.e., *coding tree blocks* are further partitioned into smaller *coding blocks*, the order traverses from *coding tree block* to *coding tree block* in *coding tree block raster scan* of the *picture*, and inside each *coding tree block*, which may be divided into *quadtrees* hierarchically to lower levels, the order traverses from *quadtree* to *quadtree* of a particular level in *quadtree-of-the-particular-level raster scan* of the *quadtree* of the immediately higher level.

4 Abbreviations

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

B	Bi-predictive
BLA	Broken Link Access
CABAC	Context-based Adaptive Binary Arithmetic Coding
CB	Coding Block
CBR	Constant Bit Rate
CLVS	Coded Layer-wise Video Sequence
CPB	Coded Picture Buffer
CRA	Clean Random Access
CTB	Coding Tree Block
CTU	Coding Tree Unit
CU	Coding Unit
CVS	Coded Video Sequence
CVSG	Coded Video Sequence Group

DPB	Decoded Picture Buffer
DUT	Decoder Under Test
EG	Exponential-Golomb
FIFO	First-In, First-Out
FIR	Finite Impulse Response
FL	Fixed-Length
GDR	Gradual Decoding Refresh
HRD	Hypothetical Reference Decoder
HSS	Hypothetical Stream Scheduler
I	Intra
IDR	Instantaneous Decoding Refresh
INBLD	Independent Non-Base Layer Decoding
IRAP	Intra Random Access Point
LPS	Least Probable Symbol
LSB	Least Significant Bit
MPS	Most Probable Symbol
MSB	Most Significant Bit
MVP	Motion Vector Prediction
NAL	Network Abstraction Layer
P	Predictive
PB	Prediction Block
PCM	Pulse Code Modulation
PPS	Picture Parameter Set
PU	Prediction Unit
RADL	Random Access Decodable Leading (Picture)
RASL	Random Access Skipped Leading (Picture)
RBSP	Raw Byte Sequence Payload
RPS	Reference Picture Set
SEI	Supplemental Enhancement Information
SLNR	Sub-Layer Non-Reference (Picture)
SODB	String Of Data Bits
SPS	Sequence Parameter Set
STSA	Step-wise Temporal Sub-layer Access
TB	Transform Block
TR	Truncated Rice
TSA	Temporal Sub-layer Access
TU	Transform Unit
UCS	Universal Coded Character Set
UTF	UCS Transmission Format
UUID	Universal Unique Identifier

VBR	Variable Bit Rate
VCL	Video Coding Layer
VPS	Video Parameter Set
VUI	Video Usability Information

5 Conventions

5.1 General

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

5.2 Arithmetic operators

The following arithmetic operators are defined as follows:

+	Addition
–	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	Multiplication, including matrix multiplication
x^y	Exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation.
/	Integer division with truncation of the result toward zero. For example, $7 / 4$ and $-7 / -4$ are truncated to 1 and $-7 / 4$ and $7 / -4$ are truncated to –1.
÷	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 $y > 0$.

5.3 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.

5.4 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.

5.5 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.
- ^ Bit-wise "exclusive or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.
- $x \gg y$ Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y . Bits shifted into the 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 non-negative integer values of y . Bits shifted into the LSBs as a result of the left shift have a value equal to 0.

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

5.7 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 and z being greater than y .

5.8 Mathematical functions

The following mathematical functions are defined:

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

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

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

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

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

Floor(x) the largest integer less than or equal to x . (5-6)

$$\text{GetCurrMsb}(cl, pl, pm, ml) = \begin{cases} pm + ml & ; \quad pl - cl \geq ml / 2 \\ pm - ml & ; \quad cl - pl > ml / 2 \\ pm & ; \quad \text{otherwise} \end{cases} \quad (5-7)$$

Log2(x) the base-2 logarithm of x . (5-8)

Log10(x) the base-10 logarithm of x . (5-9)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases} \quad (5-10)$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases} \quad (5-11)$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \quad (5-12)$$

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases} \quad (5-13)$$

$$\text{Sqrt}(x) = \sqrt{x} \quad (5-14)$$

$$\text{Swap}(x, y) = (y, x) \quad (5-15)$$

5.9 Order of operation precedence

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

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

Table 5-1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

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

Table 5-1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)

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

5.10 Variables, syntax elements, and tables

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

In some cases the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case 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 without mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the clause in which they are derived.

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

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

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

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

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

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

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

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

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

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

5.11 Text description of logical operations

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

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

may be described in the following manner:

- ... as follows / ... the following applies:
- If condition 0, statement 0
- Otherwise, if condition 1, statement 1
- ...
- Otherwise (informative remark on remaining condition), statement n

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

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

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

may be described in the following manner:

- ... as follows / ... the following applies:
- If all of the following conditions are true, statement 0:
 - condition 0a

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

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

```

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

```

may be described in the following manner:

```

When condition 0, statement 0
When condition 1, statement 1

```

5.12 Processes

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

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

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

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

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

6.1 Bitstream formats

This clause 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 Specification. The byte stream format is specified in Annex B.

6.2 Source, decoded, and output picture formats

This clause specifies the relationship between source and decoded pictures that is given via the bitstream.

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

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

- Luma (Y) only (monochrome).

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

For 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 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. 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	1
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	1	1

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

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

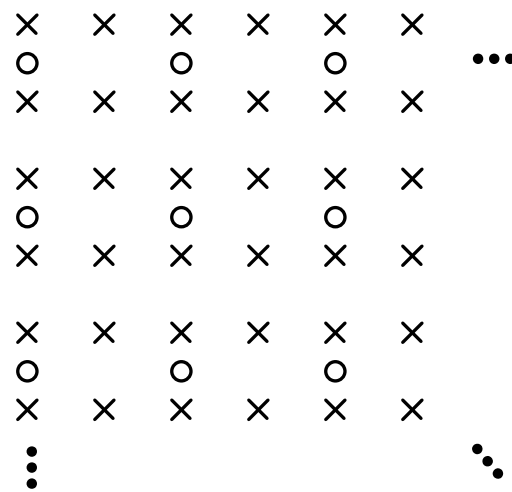
In 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array.

In 4:4:4 sampling, 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 number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence is in the range of 8 to 16, inclusive, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

When the value of chroma_format_idc is equal to 1, the nominal vertical and horizontal relative locations of luma and chroma samples in pictures are shown in Figure 6-1. Alternative chroma sample relative locations may be indicated in video usability information (see Annex E).

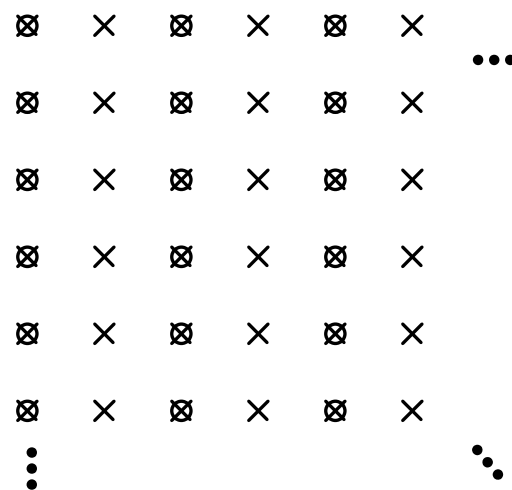


Guide:

- × = Location of luma sample
- = Location of chroma sample

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

When the value of `chroma_format_idc` is equal to 2, the chroma samples are co-sited with the corresponding luma samples and the nominal locations in a picture are as shown in Figure 6-2.



Guide:

- × = Location of luma sample
- ⊗ = Location of chroma sample

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

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

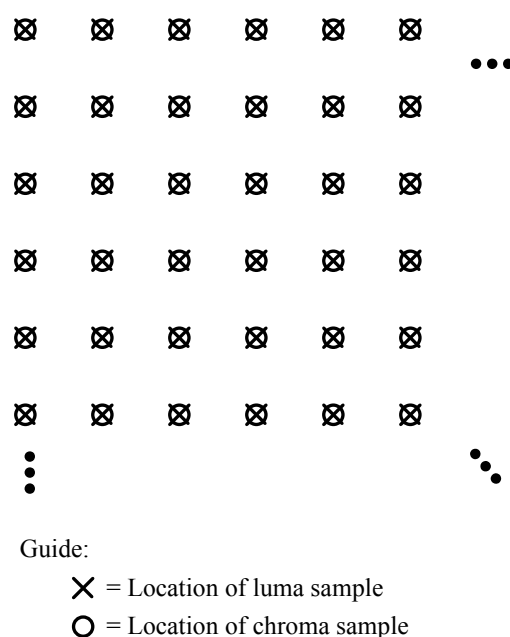


Figure 6-3 – Nominal vertical and horizontal locations of 4:4:4 luma and chroma samples in a picture

6.3 Partitioning of pictures, slices, slice segments, tiles, coding tree units, and coding tree blocks

6.3.1 Partitioning of pictures into slices, slice segments, and tiles

This clause specifies how a picture is partitioned into slices, slice segments, and tiles. Pictures are divided into slices and tiles. A slice is a sequence of one or more slice segments starting with an independent slice segment and containing all subsequent dependent slice segments (if any) that precede the next independent slice segment (if any) within the same picture. A slice segment is a sequence of coding tree units. Likewise, a tile is a sequence of coding tree units.

For example, a picture may be divided into two slices as shown in Figure 6-4. In this example, the first slice is composed of an independent slice segment containing 4 coding tree units, a dependent slice segment containing 32 coding tree units, and another dependent slice segment containing 24 coding tree units; and the second slice consists of a single independent slice segment containing the remaining 39 coding tree units of the picture.

As another example, a picture may be divided into two tiles separated by a vertical tile boundary as shown in Figure 6-5. The left side of the figure illustrates a case in which the picture only contains one slice, starting with an independent slice segment and followed by four dependent slice segments. The right side of the figure illustrates an alternative case in which the picture contains two slices in the first tile and one slice in the second tile.

Unlike slices, tiles are always rectangular. A tile always contains an integer number of coding tree units, and may consist of coding tree units contained in more than one slice. Similarly, a slice may consist of coding tree units contained in more than one tile.

One or both of the following conditions shall be fulfilled for each slice and tile:

- All coding tree units in a slice belong to the same tile.
- All coding tree units in a tile belong to the same slice.

NOTE 1 – Within the same picture, there may be both slices that contain multiple tiles and tiles that contain multiple slices.

One or both of the following conditions shall be fulfilled for each slice segment and tile:

- All coding tree units in a slice segment belong to the same tile.
- All coding tree units in a tile belong to the same slice segment.

When a picture is coded using three separate colour planes (separate_colour_plane_flag is equal to 1), a slice contains only coding tree blocks 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 a picture may be interleaved with each other under the constraint that for each value of colour_plane_id, the coded slice segment NAL units with that value of colour_plane_id shall be in the order of increasing coding tree block address in tile scan order for the first coding tree block of each coded slice segment NAL unit.

NOTE 2 – When `separate_colour_plane_flag` is equal to 0, each coding tree block of a picture is contained in exactly one slice. When `separate_colour_plane_flag` is equal to 1, each coding tree block of a colour component is contained in exactly one slice (i.e., information for each coding tree block of a picture is present in exactly three slices and these three slices have different values of `colour_plane_id`).

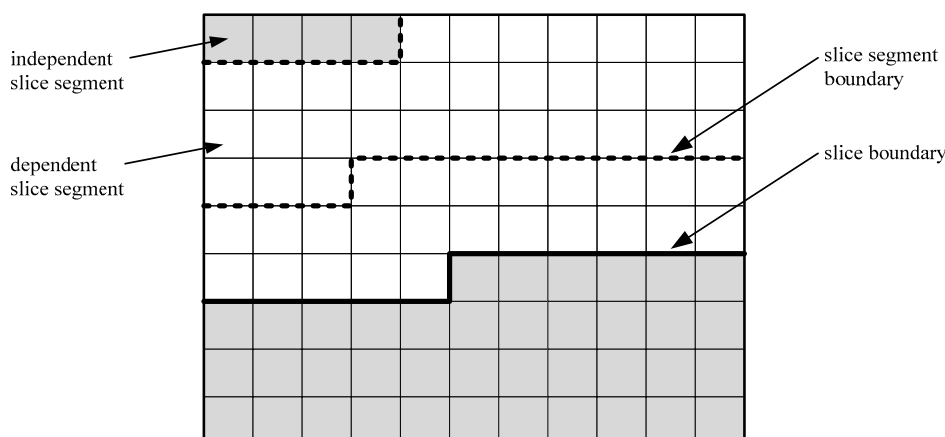


Figure 6-4 – A picture with 11 by 9 luma coding tree blocks that is partitioned into two slices, the first of which is partitioned into three slice segments (informative)

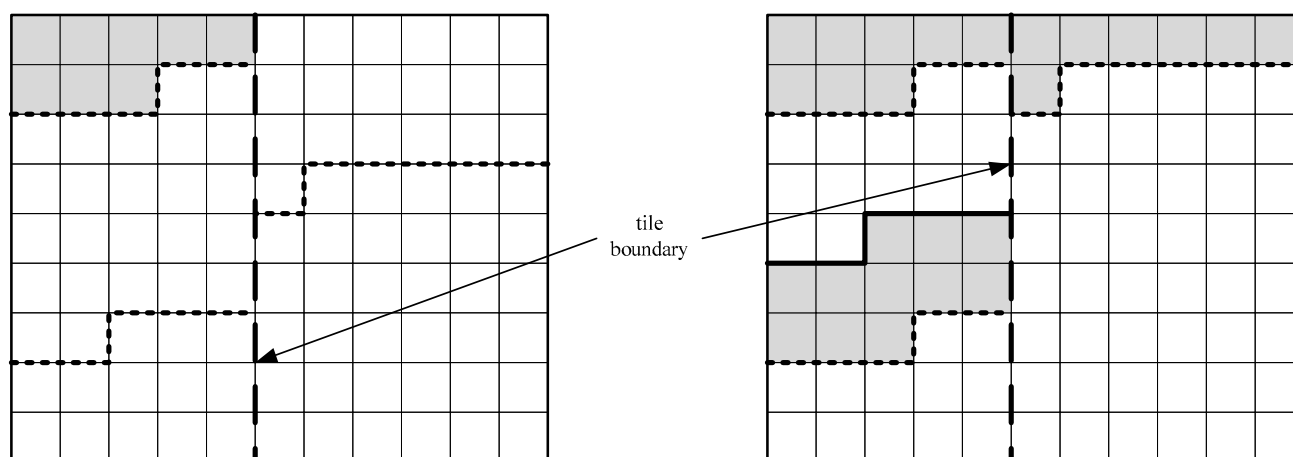


Figure 6-5 – A picture with 11 by 9 luma coding tree blocks that is partitioned into two tiles and one slice (left) or is partitioned into two tiles and three slices (right) (informative)

6.3.2 Block and quadtree structures

The samples are processed in units of coding tree blocks. The array size for each luma coding tree block in both width and height is `CtbSizeY` in units of samples. The width and height of the array for each chroma coding tree block are `CtbWidthC` and `CtbHeightC`, respectively, in units of samples.

Each coding tree block is assigned a partition signalling to identify the block sizes for intra or inter prediction and for transform coding. The partitioning is a recursive quadtree partitioning. The root of the quadtree is associated with the coding tree block. The quadtree is split until a leaf is reached, which is referred to as the coding block. When the component width is not an integer number of the coding tree block size, the coding tree blocks at the right component boundary are incomplete. When the component height is not an integer multiple of the coding tree block size, the coding tree blocks at the bottom component boundary are incomplete.

The coding block is the root node of two trees, the prediction tree and the transform tree. The prediction tree specifies the position and size of prediction blocks. The transform tree specifies the position and size of transform blocks. The splitting information for luma and chroma is identical for the prediction tree and may or may not be identical for the transform tree.

The blocks and associated syntax structures are grouped into "unit" structures as follows:

- One prediction block (monochrome picture or `separate_colour_plane_flag` is equal to 1) or three prediction blocks (luma and chroma components of a picture in 4:2:0 or 4:4:4 colour format) or five prediction blocks (luma and chroma components of a picture in 4:2:2 colour format) and the associated prediction syntax structures units are associated with a prediction unit.
- One transform block (monochrome picture or `separate_colour_plane_flag` is equal to 1) or three transform blocks (luma and chroma components of a picture in 4:2:0 or 4:4:4 colour format) or five transform blocks (luma and chroma components of a picture in 4:2:2 colour format) and the associated transform syntax structures units are associated with a transform unit.
- One coding block (monochrome picture or `separate_colour_plane_flag` is equal to 1) or three coding blocks (luma and chroma), the associated coding syntax structures and the associated prediction and transform units are associated with a coding unit.
- One coding tree block (monochrome picture or `separate_colour_plane_flag` is equal to 1) or three coding tree blocks (luma and chroma), the associated coding tree syntax structures and the associated coding units are associated with a coding tree unit.

6.3.3 Spatial or component-wise partitionings

The following divisions of processing elements of this Specification form spatial or component-wise partitionings:

- The division of each picture into components
- The division of each component into coding tree blocks
- The division of each picture into tile columns
- The division of each picture into tile rows
- The division of each tile column into tiles
- The division of each tile row into tiles
- The division of each tile into coding tree units
- The division of each picture into slices
- The division of each slice into slice segments
- The division of each slice segment into coding tree units
- The division of each coding tree unit into coding tree blocks
- The division of each coding tree block into coding blocks, except that the coding tree blocks are incomplete at the right component boundary when the component width is not an integer multiple of the coding tree block size and the coding tree blocks are incomplete at the bottom component boundary when the component height is not an integer multiple of the coding tree block size
- The division of each coding tree unit into coding units, except that the coding tree units are incomplete at the right picture boundary when the picture width in luma samples is not an integer multiple of the luma coding tree block size and the coding tree units are incomplete at the bottom picture boundary when the picture height in luma samples is not an integer multiple of the luma coding tree block size
- The division of each coding unit into prediction units
- The division of each coding unit into transform units
- The division of each coding unit into coding blocks
- The division of each coding block into prediction blocks
- The division of each coding block into transform blocks
- The division of each prediction unit into prediction blocks
- The division of each transform unit into transform blocks.

6.4 Availability processes

6.4.1 Derivation process for z-scan order block availability

Inputs to this process are:

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

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

The minimum luma block address in z-scan order $minBlockAddrCurr$ of the current block is derived as follows:

$$minBlockAddrCurr = MinTbAddrZs[x_{Curr} \gg MinTbLog2SizeY][y_{Curr} \gg MinTbLog2SizeY] \quad (6-1)$$

The minimum luma block address in z-scan order $minBlockAddrN$ of the neighbouring block covering the location (x_{NbY} , y_{NbY}) is derived as follows:

- If one or more of the following conditions are true, $minBlockAddrN$ is set equal to -1 :
 - x_{NbY} is less than 0
 - y_{NbY} is less than 0
 - x_{NbY} is greater than or equal to $pic_width_in_luma_samples$
 - y_{NbY} is greater than or equal to $pic_height_in_luma_samples$
- Otherwise (x_{NbY} and y_{NbY} are inside the picture boundaries),

$$minBlockAddrN = MinTbAddrZs[x_{NbY} \gg MinTbLog2SizeY][y_{NbY} \gg MinTbLog2SizeY] \quad (6-2)$$

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

- If one or more of the following conditions are true, $availableN$ is set equal to FALSE:
 - $minBlockAddrN$ is less than 0,
 - $minBlockAddrN$ is greater than $minBlockAddrCurr$,
 - the variable $SliceAddrRs$ associated with the slice segment containing the neighbouring block with the minimum luma block address $minBlockAddrN$ differs in value from the variable $SliceAddrRs$ associated with the slice segment containing the current block with the minimum luma block address $minBlockAddrCurr$.
 - the neighbouring block with the minimum luma block address $minBlockAddrN$ is contained in a different tile than the current block with the minimum luma block address $minBlockAddrCurr$.
- Otherwise, $availableN$ is set equal to TRUE.

6.4.2 Derivation process for prediction block availability

Inputs to this process are:

- the luma location (x_{Cb} , y_{Cb}) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable $nCbS$ specifying the size of the current luma coding block,
- the luma location (x_{Pb} , y_{Pb}) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
- two variables $nPbW$ and $nPbH$ specifying the width and the height of the current luma prediction block,
- a variable $partIdx$ specifying the partition index of the current prediction unit within the current coding unit,
- the luma location (x_{NbY} , y_{NbY}) covered by a neighbouring prediction block relative to the top-left luma sample of the current picture.

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

The variable `sameCb` specifying whether the current luma prediction block and the neighbouring luma prediction block cover the same luma coding block.

- If all of the following conditions are true, `sameCb` is set equal to `TRUE`:
 - `xCb` is less than or equal than `xNbY`,
 - `yCb` is less than or equal than `yNbY`,
 - $(xCb + nCbS)$ is greater than `xNbY`,
 - $(yCb + nCbS)$ is greater than `yNbY`.
- Otherwise, `sameCb` is set equal to `FALSE`.

The neighbouring prediction block availability `availableN` is derived as follows:

- If `sameCb` is equal to `FALSE`, the derivation process for z-scan order block availability as specified in clause 6.4.1 is invoked with $(xCurr, yCurr)$ set equal to (xPb, yPb) and the luma location $(xNbY, yNbY)$ as inputs, and the output is assigned to `availableN`.
- Otherwise, if all of the following conditions are true, `availableN` is set equal to `FALSE`:
 - $(nPbW \ll 1)$ is equal to `nCbS`,
 - $(nPbH \ll 1)$ is equal to `nCbS`,
 - `partIdx` is equal to 1,
 - $(yCb + nPbH)$ is less than or equal to `yNbY`,
 - $(xCb + nPbW)$ is greater than `xNbY`.
- Otherwise, `availableN` is set equal to `TRUE`.

When `availableN` is equal to `TRUE` and `CuPredMode[xNbY][yNbY]` is equal to `MODE_INTRA`, `availableN` is set equal to `FALSE`.

6.5 Scanning processes

6.5.1 Coding tree block raster and tile scanning conversion process

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

```

if( uniform_spacing_flag )
    for( i = 0; i <= num_tile_columns_minus1; i++ )
        colWidth[ i ] = ( ( i + 1 ) * PicWidthInCtbsY ) / ( num_tile_columns_minus1 + 1 ) -
            ( i * PicWidthInCtbsY ) / ( num_tile_columns_minus1 + 1 )
else {
    colWidth[ num_tile_columns_minus1 ] = PicWidthInCtbsY
    for( i = 0; i < num_tile_columns_minus1; i++ ) {
        colWidth[ i ] = colWidth[ i + 1 ] + 1
        colWidth[ num_tile_columns_minus1 ] -= colWidth[ i ]
    }
}

```

(6-3)

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

```

if( uniform_spacing_flag )
    for( j = 0; j <= num_tile_rows_minus1; j++ )
        rowHeight[ j ] = ( ( j + 1 ) * PicHeightInCtbsY ) / ( num_tile_rows_minus1 + 1 ) -
            ( j * PicHeightInCtbsY ) / ( num_tile_rows_minus1 + 1 )
else {
    rowHeight[ num_tile_rows_minus1 ] = PicHeightInCtbsY
    for( j = 0; j < num_tile_rows_minus1; j++ ) {
        rowHeight[ j ] = rowHeight[ j + 1 ] + 1
        rowHeight[ num_tile_rows_minus1 ] -= rowHeight[ j ]
    }
}

```

(6-4)

The list $\text{colBd}[i]$ for i ranging from 0 to $\text{num_tile_columns_minus1} + 1$, inclusive, specifying the location of the i -th tile column boundary in units of coding tree blocks, is derived as follows:

```
for( colBd[ 0 ] = 0, i = 0; i <= num_tile_columns_minus1; i++ )
    colBd[ i + 1 ] = colBd[ i ] + colWidth[ i ]
```

 (6-5)

The list $\text{rowBd}[j]$ for j ranging from 0 to $\text{num_tile_rows_minus1} + 1$, inclusive, specifying the location of the j -th tile row boundary in units of coding tree blocks, is derived as follows:

```
for( rowBd[ 0 ] = 0, j = 0; j <= num_tile_rows_minus1; j++ )
    rowBd[ j + 1 ] = rowBd[ j ] + rowHeight[ j ]
```

 (6-6)

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

```
for( ctbAddrRs = 0; ctbAddrRs < PicSizeInCtbsY; ctbAddrRs++ ) {
    tbX = ctbAddrRs % PicWidthInCtbsY
    tbY = ctbAddrRs / PicWidthInCtbsY
    for( i = 0; i <= num_tile_columns_minus1; i++ )
        if( tbX >= colBd[ i ] )
            tileX = i
    for( j = 0; j <= num_tile_rows_minus1; j++ )
        if( tbY >= rowBd[ j ] )
            tileY = j
    CtbAddrRsToTs[ ctbAddrRs ] = 0
    for( i = 0; i < tileX; i++ )
        CtbAddrRsToTs[ ctbAddrRs ] += rowHeight[ tileY ] * colWidth[ i ]
    for( j = 0; j < tileY; j++ )
        CtbAddrRsToTs[ ctbAddrRs ] += PicWidthInCtbsY * rowHeight[ j ]
    CtbAddrRsToTs[ ctbAddrRs ] += ( tbY - rowBd[ tileY ] ) * colWidth[ tileX ] + tbX - colBd[ tileX ]
}
```

 (6-7)

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

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

 (6-8)

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

```
for( j = 0, tileIdx = 0; j <= num_tile_rows_minus1; j++ )
    for( i = 0; i <= num_tile_columns_minus1; i++, tileIdx++ )
        for( y = rowBd[ j ]; y < rowBd[ j + 1 ]; y++ )
            for( x = colBd[ i ]; x < colBd[ i + 1 ]; x++ )
                TileId[ CtbAddrRsToTs[ y * PicWidthInCtbsY + x ] ] = tileIdx
```

 (6-9)

The values of $\text{ColumnWidthInLumaSamples}[i]$, specifying the width of the i -th tile column in units of luma samples, are set equal to $\text{colWidth}[i] \ll \text{CtbLog2SizeY}$ for i ranging from 0 to $\text{num_tile_columns_minus1}$, inclusive.

The values of $\text{RowHeightInLumaSamples}[j]$, specifying the height of the j -th tile row in units of luma samples, are set equal to $\text{rowHeight}[j] \ll \text{CtbLog2SizeY}$ for j ranging from 0 to $\text{num_tile_rows_minus1}$, inclusive.

6.5.2 Z-scan order array initialization process

The array MinTbAddrZs with elements $\text{MinTbAddrZs}[x][y]$ for x ranging from 0 to $(\text{PicWidthInCtbsY} \ll (\text{CtbLog2SizeY} - \text{MinTbLog2SizeY})) - 1$, inclusive, and y ranging from 0 to $(\text{PicHeightInCtbsY} \ll (\text{CtbLog2SizeY} - \text{MinTbLog2SizeY})) - 1$, specifying the conversion from a location (x, y) in units of minimum blocks to a minimum block address in z-scan order, inclusive is derived as follows:

```
for( y = 0; y < ( PicHeightInCtbsY << ( CtbLog2SizeY - MinTbLog2SizeY ) ); y++ )
    for( x = 0; x < ( PicWidthInCtbsY << ( CtbLog2SizeY - MinTbLog2SizeY ) ); x++ ) {
        tbX = ( x << MinTbLog2SizeY ) >> CtbLog2SizeY
        tbY = ( y << MinTbLog2SizeY ) >> CtbLog2SizeY
        ctbAddrRs = PicWidthInCtbsY * tbY + tbX
        MinTbAddrZs[ x ][ y ] = CtbAddrRsToTs[ ctbAddrRs ] <<
            ( ( CtbLog2SizeY - MinTbLog2SizeY ) * 2 )
        for( i = 0, p = 0; i < ( CtbLog2SizeY - MinTbLog2SizeY ); i++ ) {
            m = 1 << i
```

 (6-10)

```

        p += ( m & x ? m * m : 0 ) + ( m & y ? 2 * m * m : 0 )
    }
    MinTbAddrZs[ x ][ y ] += p
}

```

6.5.3 Up-right diagonal scan order array initialization process

Input to this process is a block size blkSize.

Output of this process is the array diagScan[sPos][sComp]. The array index sPos specify the scan position ranging from 0 to (blkSize * blkSize) – 1. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. Depending on the value of blkSize, the array diagScan is derived as follows:

```

i = 0
x = 0
y = 0
stopLoop = FALSE
while( !stopLoop ) {
    while( y >= 0 ) {
        if( x < blkSize && y < blkSize ) {
            diagScan[ i ][ 0 ] = x
            diagScan[ i ][ 1 ] = y
            i++
        }
        y--
        x++
    }
    y = x
    x = 0
    if( i >= blkSize * blkSize )
        stopLoop = TRUE
}

```

(6-11)

6.5.4 Horizontal scan order array initialization process

Input to this process is a block size blkSize.

Output of this process is the array horScan[sPos][sComp]. The array index sPos specifies the scan position ranging from 0 to (blkSize * blkSize) – 1. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. Depending on the value of blkSize, the array horScan is derived as follows:

```

i = 0
for( y = 0; y < blkSize; y++ )
    for( x = 0; x < blkSize; x++ ) {
        horScan[ i ][ 0 ] = x
        horScan[ i ][ 1 ] = y
        i++
    }

```

(6-12)

6.5.5 Vertical scan order array initialization process

Input to this process is a block size blkSize.

Output of this process is the array verScan[sPos][sComp]. The array index sPos specifies the scan position ranging from 0 to (blkSize * blkSize) – 1. The array index sComp equal to 0 specifies the horizontal component and the array index sComp equal to 1 specifies the vertical component. Depending on the value of blkSize, the array verScan is derived as follows:

```

i = 0
for( x = 0; x < blkSize; x++ )
    for( y = 0; y < blkSize; y++ ) {
        verScan[ i ][ 0 ] = x
        verScan[ i ][ 1 ] = y
        i++
    }

```

(6-13)

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 some means for identifying entry points into the bitstream and some means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified in this Specification.

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

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

7.2 Specification of syntax functions and descriptors

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

`byte_aligned()` is specified as follows:

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

`more_data_in_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_data_in_payload()` is specified as follows:

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

`more_rbsp_data()` is specified as follows:

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

The method for enabling determination of whether there is more data in the RBSP is specified by the application (or in Annex B for applications that use the 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.

`payload_extension_present()` is specified as follows:

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

`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 format as specified in Annex B and fewer than `n` bits remain within the byte stream, `next_bits( n )` returns a value of 0.

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

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

- `ae(v)`: context-adaptive arithmetic entropy-coded syntax element. The parsing process for this descriptor is specified in clause 9.3.
- `b(8)`: byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function `read_bits( 8 )`.
- `f(n)`: fixed-pattern bit string using `n` bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )`.
- `i(n)`: signed integer using `n` bits. When `n` is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )` interpreted as a two's complement integer representation with most significant bit written first.
- `se(v)`: signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in clause 9.2.
- `st(v)`: null-terminated string encoded as UTF-8 characters as specified in ISO/IEC 10646. The parsing process is specified as follows: `st(v)` begins at a byte-aligned position in the bitstream and reads and returns a series of bytes from the bitstream, beginning at the current position and continuing up to but not including the next byte-aligned byte that is equal to 0x00, and advances the bitstream pointer by $(\text{stringLength} + 1) * 8$ bit positions, where `stringLength` is equal to the number of bytes returned.
NOTE – The `st(v)` syntax descriptor is only used in this Specification when the current position in the bitstream is a byte-aligned position.
- `u(n)`: unsigned integer using `n` bits. When `n` is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function `read_bits( n )` interpreted as a binary representation of an unsigned integer with most significant bit written first.
- `ue(v)`: unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in clause 9.2.

7.3 Syntax in tabular form

7.3.1 NAL unit syntax

7.3.1.1 General NAL unit syntax

	Descriptor
<code>nal_unit(NumBytesInNalUnit) {</code>	
<code>nal_unit_header()</code>	
<code>NumBytesInRbsp = 0</code>	
<code>for(i = 2; i < NumBytesInNalUnit; i++)</code>	
<code>if(i + 2 < NumBytesInNalUnit && next_bits(24) == 0x000003) {</code>	
<code>rbsp_byte[NumBytesInRbsp++]</code>	b(8)
<code>rbsp_byte[NumBytesInRbsp++]</code>	b(8)
<code>i += 2</code>	
<code>emulation_prevention_three_byte /* equal to 0x03 */</code>	f(8)
<code>} else</code>	
<code>rbsp_byte[NumBytesInRbsp++]</code>	b(8)
<code>}</code>	

7.3.1.2 NAL unit header syntax

<code>nal_unit_header()</code> {	Descriptor
forbidden_zero_bit	f(1)
nal_unit_type	u(6)
nuh_layer_id	u(6)
nuh_temporal_id_plus1	u(3)
}	