INTERNATIONAL STANDARD

ISO/IEC 21122-1

Second edition
2022-03

Information technology — JPEG XS low-latency lightweight image coding system —

Part 1: Core coding system

Technologies de l'information — Système de codage d'images léger à faible latence JPEG XS —

Partie 1: Système de codage de noyau

Reference number
ISO/IEC 21122-1:2022(E)

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

Published in Switzerland

Contents

Page

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

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

This second edition cancels and replaces the first edition (ISO/IEC 21122-1:2019), which has been technically revised.

The main changes are as follows:

— coding tools for compressing colour filter array images (CFA images) have been added;

— coding tools that enable lossless coding of images with up to 12 bits per sample have been added;

— support for 4:2:0 sampled images has been added.

A list of all parts in the ISO/IEC 21122 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

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

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

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

Information technology — JPEG XS low-latency lightweight image coding system —

Part 1: Core coding system

1 Scope

This document defines the syntax and an accompanying decompression process that is capable to represent continuous-tone grey-scale, or continuous-tone colour digital images without visual loss at moderate compression rates. Typical compression rates are between 2:1 and 6:1 but can also be higher depending on the nature of the image. In particular, the syntax and the decoding process specified in this document allow lightweight encoder and decoder implementations that limit the end-to-end latency to a fraction of the frame size. However, the definition of transmission channel buffer models necessary to ensure such latency is beyond the scope of this document.

This document:

— specifies decoding processes for converting compressed image data to reconstructed image data;

— specifies a codestream syntax containing information for interpreting the compressed image data;

— provides guidance on encoding processes for converting source image data to compressed image data.

2 Normative references

There are no normative references in this document.

3 Terms and definitions, abbreviated terms and symbols

3.1 Terms and definitions

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

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

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

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

3.1.1
band
input data to a specific *wavelet filter type* (3.1.58) that contributes to the generation of one of the *components* (3.1.14) of the image

3.1.2
band type
single number collapsing the information on the component, and horizontal and vertical wavelet filter types that are applied in the filter cascade reconstructing spatial image samples from inversely quantized wavelet coefficients

3.1.3
bit
binary choice encoded as either 0 or 1

3.1.4
bitplane
array of bits having all the same significance

3.1.5
bitplane count
number of significant bitplanes of a code group, counting from the LSB up to the most significant, non-empty bitplane

3.1.6
bitplane count subpacket
subset of a packet which decodes to the bitplane counts of all code groups within a packet, followed by padding and optional filler bytes

Note 1 to entry: See subclause C.5.3.

3.1.7
byte
group of 8 bits

3.1.8
colour filter array
CFA
rectangular array of sensor elements yielding a 1-component picture where the colour to which a sensor element is sensitive to depends on the position of the sensor element

3.1.9
codestream
compressed image data representation that includes all necessary data to allow a (full or approximate) reconstruction of the sample values of a digital image

3.1.10
code group
group of quantization indices in sign-magnitude representation before inverse quantization

3.1.11
coefficient
input value to the inverse wavelet transformation resulting from inverse quantization

3.1.12
coefficient group
number of horizontally adjacent wavelet coefficients from the same band

3.1.13
column
set of vertically aligned precincts

3.1.14
component
two-dimensional array of samples having the same designation such as red, green or blue in the output or display device

3.1.15
compression
process of reducing the number of bits used to represent source image data

3.1.16
continuous-tone image
image whose components have more than one bit per sample

3.1.17
data subpacket
subset of a packet which consists of the quantization index magnitudes, followed by padding and optional filler bytes

Note 1 to entry: See subclause C.5.4.

3.1.18
deadzone quantizer
quantizer whose zero bucket has a size different from all other buckets

3.1.19
decoder
embodiment of a decoding process

3.1.20
decoding process
process which takes as its input a codestream and outputs a continuous-tone image

3.1.21
decomposition level
set of wavelet coefficients resulting from a particular level of recursive application of a wavelet transform

3.1.22
downsampling
procedure by which the spatial resolution of a component is reduced

3.1.23
encoder
embodiment of an encoding process

3.1.24
encoding process
process which outputs compressed image data in the form of a codestream

3.1.25
entropy decoder
embodiment of an entropy decoding procedure

3.1.26
entropy decoding
lossless procedure which recovers the sequence of symbols from the sequence of bits produced by the entropy encoder

3.1.27
entropy encoder
embodiment of an entropy encoding procedure

3.1.28
entropy encoding
lossless procedure which converts a sequence of input symbols into a sequence of bits such that the average number of bits per symbol approaches the entropy of the input symbols

3.1.29
filler bytes
integer number of bytes a decoder will skip over on decoding without interpreting the values of the bytes itself

3.1.30
grayscale image
continuous-tone image that has only one component

3.1.31
inverse quantization
inverse procedure to quantization by which the decoder recovers a representation of the coefficients

3.1.32
inverse reversible multi component transformation
inverse RCT
inverse transformation across multiple component sample values located at the same sample grid point that is invertible without loss

Note 1 to entry: See subclauses F.3 and F.4.

3.1.33
LL band
input to a series of wavelet filters where only inverse low-pass filters are applied in horizontal and vertical direction

3.1.34
lossless
descriptive term for encoding and decoding processes and procedures in which the output of the decoding procedure(s) is identical to the input to the encoding procedure(s)

3.1.35
lossless coding
mode of operation which refers to any one of the coding processes defined in this document in which all of the procedures are lossless

3.1.36
lossy
descriptive term for encoding and decoding processes which are not lossless

3.1.37
packet
segment of the codestream containing entropy coded information on a single precinct, line and a subset of the bands within this precinct and line

3.1.38
padding
bits within the codestream whose only purpose is to align syntax elements to byte boundaries and that carry no information

3.1.39
precinct
collection of quantization indices of all bands contributing to a given spatial region of the image

3.1.40
precision
number of bits allocated to a particular sample, coefficient, or other binary numerical representation

3.1.41
procedure
set of steps which accomplishes one of the tasks which comprise an encoding or decoding process

3.1.42
quantization
method of reducing the precision of the individual coefficients

3.1.43
quantization index
input to the inverse quantization process which reconstructs the quantization index to a wavelet coefficient

3.1.44
quantization index magnitude
absolute value of a quantization index

3.1.45
sample
one element in the two-dimensional image array which comprises a component

3.1.46
sample grid
common coordinate system for all samples of an image, the samples at the top left edge of the image have the coordinates (0,0), the first coordinate increases towards the right, the second towards the bottom

3.1.47
sign subpacket
subset of a packet that consists of the sign information of all non-zero quantization indices within a packet, followed by padding and optional filler bytes

Note 1 to entry: See subclause C.5.5.

3.1.48
significance
attribute of code groups that applies if, depending on the Run Mode flag in the picture header, either at least one of coefficients in the code group is non-zero, or the bitplane count prediction residual of the code group is non-zero

3.1.49
significance group
group of a horizontally adjacent code groups sharing the same significance information in the significance subpacket

3.1.50
significance subpacket
subset of a packet that identifies which significance groups within a packet are insignificant, followed by padding and optional filler bytes

Note 1 to entry: see subclause C.5.2

3.1.51
slice
integral number of precincts whose wavelet coefficients can be entropy-decoded independently

3.1.52
star-tetrix
decorrelation transformation that combines a spatial with an inter-component decorrelation transformation particularly tuned for CFA pattern compression

Note 1 to entry: see subclause F.5

3.1.53
subpacket
substructure of a packet containing information of one or multiple bands of one line of a single precinct

3.1.54
super pixel
2×2 arrangement of sensor elements in a CFA pattern array containing at least one sensor element for each colour filter type

3.1.55
truncation position
number of least significant bitplanes not included in the quantization index of a wavelet coefficient

3.1.56
uniform quantizer
quantizer whose buckets are all of equal size

3.1.57
upsampling
procedure by which the spatial resolution of a component is increased

3.1.58
wavelet filter type
single number that uniquely identifies each element of the wavelet filter with regard to the number and type of horizontal and vertical decompositions

Note 1 to entry: Unlike the band type, the wavelet filter type does not include component information.

3.2 Abbreviated terms

JPEG XS	informal name of this standard where XS stands for “extra speed”
LSB	least significant bit
MSB	most significant bit

3.3 Symbols

$B[c]$	bit precision of component c
β	wavelet filter type
b	band type
$b_x[\beta,i]$	band existence flag for filter type β in component i. 1 if the filter exists, 0 otherwise.
$b'_x[b]$	band existence flag for band type b. 1 if the filter exists, 0 otherwise.
Bw	nominal overall bit precision of the wavelet data
B_r	number of bits required to encode a bitplane count in raw
$Cpih$	colour transformation type
$c[p,\lambda,b,x]$	wavelet coefficient in precinct p, line λ, band b and position x
C_s	width of precincts other than the rightmost precinct in sample grid positions
C_t	colour transformation CFA pattern type derived from the component registration

C_f	colour transformation reflection and extension flags
Cw	width of precincts in multiples of 8 LL subsampled band sample grid positions
$D[p,b]$	bitplane count coding mode of band b in precinct p
$D_r[p,s]$	raw coding mode override flag for packet s in precinct p
DCO	DC offset
$d_x[\beta,i]$	horizontal decomposition level of wavelet filter type β of component i
$d_y[\beta,i]$	vertical decomposition level of wavelet filter type β of component i
$\delta_x[c]$	horizontal position of component c in a CFA super pixel
$\delta_y[c]$	vertical position of component c in a CFA super pixel
E	exponent of the slope of the linear region of the extended non-linearity
e_1	colour transformation exponent of first chroma component
e_2	colour transformation exponent of second chroma component
Fs	sign packing flag
$Fslc$	slice coding mode
Fq	number of fractional bits in the representation of wavelet coefficients
$G[b]$	gain of subband b
$H_b[\beta,k]$	height of filter type β of component k in wavelet coefficients
$H_c[i]$	height of the component i in sample points
H_f	height of the image in sampling grid points
H_p	height of a precinct in lines
H_{sl}	height of a slice in precincts
$I[p,b,\lambda,s]$	line inclusion flag, set if line λ of band b and precinct p is included in packet s, reset otherwise
$k[\delta_x, \delta_y]$	Component within CFA super pixel at position δ_x, δ_y
$L_0[p,b]$	first line of band b in precinct p
$L_1[p,b]$	last line + 1 of band b in precinct p
$Lcod$	codestream length in bytes
$L_{cnt}[p,s]$	size of the bitplane count subpacket of precinct p and packet s in bytes
$L_{dat}[p,s]$	size of the data subpacket of precinct p and packet s in bytes
Lh	long header flag in in the picture header, set if long headers are enforced, reset otherwise
$L_{prc}[p]$	length of the entropy coded data in precinct p
$L_{sgn}[p,s]$	size of the sign subpacket of precinct p and packet s in bytes

$L_{sig}[p,s]$	size of the significance subpacket of precinct p and packet s in bytes
$M[p,\lambda,b,g]$	bitplane count of precinct p, line λ, band b and code group g
$M_{top}[p,\lambda,b,g]$	vertical predictor of the bitplane count of precinct p, line λ, band b and code group g
N_c	number of components in an image
$N_{cg}[p,b]$	number of code groups in precinct p and band b
N_β	number of bands per component
N_g	number of coefficients in a code group
$N_s[p,b]$	number of significance groups per line band b of precinct p
$N_p[t]$	number of precincts in slice t
N_L	number of bands in the wavelet decomposition of the image (wavelet filter types times components)
$N_{L,x}$	maximal number of horizontal decomposition levels
$N'_{L,x}[i]$	number of horizontal decomposition levels of component i
$N_{L,y}$	maximal number of vertical decomposition levels over all components
$N'_{L,y}[i]$	number of vertical decomposition levels of component i
$N_{p,x}$	number of precincts per sampling grid line
$N_{p,y}$	number of precincts per sampling grid column
$N_{pc}[p]$	number of packets in precinct p
$O[c,x,y]$	unscaled output of the inverse wavelet transformation at coordinates x and y of the component c
$\Omega[c,x,y]$	output of the inverse multiple component transformation at position x,y for component c
$P[b]$	priority of band b
Plev	level a particular codestream complies to
Ppih	profile a particular codestream complies to
Ppoc	progression order in which bands are transmitted in the codestream
$Q[p]$	quantization parameter of precinct p
Qpih	quantization type of the picture
Rl	raw-mode selection per packet flag
Rm	run mode used for significance coding
$R[p]$	refinement of precinct p
$R[c,x,y]$	reconstructed sample value at position x,y for component c
Sd	number of components for which wavelet decomposition is suppressed

S_s	size of a significance group in code groups
$s_x[i]$	sampling factor of component *i* in horizontal direction
$s_y[i]$	sampling factor of component *i* in vertical direction
$s[p,\lambda,b,x]$	sign of the wavelet coefficient in precinct *p*, line λ, band *b* and position *x*.
$T1$	first threshold of the extended non-linearity
$T2$	second threshold of the extended non-linearity
$T[p,b]$	truncation position of precinct *p* and band *b*
$T_{top}[p,b]$	vertical Truncation position predictor of precinct *p* and band *b*
$T[\beta,x,y]$	temporary wavelet coefficient of filter type β at location *x,y*.
$v[x,y]$	sample value at the sample grid position *x,y*
$v[p,\lambda,b,x]$	quantization index magnitude of the wavelet coefficient in precinct *p*, line λ, band *b* and position *x*
$W_b[\beta,k]$	width of filter type β of component *k* in wavelet coefficients
$W_c[i]$	width of component *i* in samples
W_f	width of the image in sampling grid points
$W_p[p]$	width of the precinct *p* in sampling grid points
$W_{pb}[p,b]$	width of subband b of precinct p in coefficients
Wt_x	wavelet filter type for horizontal filtering
Wt_y	wavelet filter type for vertical filtering
$X[y]$	one-dimensional temporal array of wavelet coefficients
$Xcrg[c]$	horizontal component registration of component *c* relative to the sample grid
$Ycrg[c]$	vertical component registration of component *c* relative to the sample grid
Yslh	vertical slice order within the picture
$Z[p,\lambda,b,j]$	significance flag of precinct *p*, line λ, band *b* and significance group *j*

4 Conventions

4.1 Conformance language

The keyword "reserved" indicates a provision that is not specified at this time, shall not be used, and may be specified in the future. The keyword "forbidden" indicates "reserved" and in addition indicates that the provision will never be specified in the future.

4.2 Operators

NOTE Many of the operators used in document are similar to those used in the C programming language.

4.2.1 Arithmetic operators

&	bitwise AND operation
+	addition
–	subtraction (as a binary operator) or negation (as a unary prefix operator)
×	multiplication
/	division without truncation or rounding
<<	left shift: $x<<s$ is defined as $x \times 2^s$
>>	right shift: $x>>s$ is defined as $\lfloor x/2^s \rfloor$
umod	x umod a is the unique value y between 0 and $a-1$ for which $y+Na = x$ with a suitable integer N

4.2.2 Logical operators

||	logical OR
&&	logical AND
!	logical NOT

4.2.3 Relational operators

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

4.2.4 Precedence order of operators

NOTE Operators are listed below in descending order of precedence. If several operators appear in the same line, they have equal precedence. When several operators of equal precedence appear at the same level in an expression, evaluation proceeds according to the associativity of the operator either from right to left or from left to right.

Operators	Type of operation	Associativity
()	expression	left to right
[]	indexing of arrays	left to right
–	unary negation	
×, /	multiplication, division	left to right
mod	modulo (remainder)	left to right
+, –	addition and subtraction	left to right

<<, >>	left shift and right shift	left to right
< , >, ≤, ≥	relational	left to right
&	bitwise AND	left to right

4.2.5 Mathematical functions

$\lceil x \rceil$	ceil of x: returns the smallest integer that is greater than or equal to x
$\lfloor x \rfloor$	floor of x: returns the largest integer that is less than or equal to x
$\lvert x \rvert$	absolute value of x, $\lvert x \rvert$ equals $-x$ for $x < 0$, otherwise x
$\log_2(x)$	logarithm to the basis of 2, e.g. $\log_2(1)=0$ and $\log_2(2)=1$
sign(x)	sign of x, 0 if x is 0, +1 if x is positive, -1 if x is negative
$\sqrt{x}$	square root of x, i.e. non-negative number y such that $y \times y = x$
clamp(x,min,max)	clamp x to the range [min,max] clamp(x,min,max) equals min if $x < min$, max if $x > max$ or otherwise x
$\max_i(x_i)$	maximum of a sequence of numbers $\{x_i\}$ enumerated by the index i
max(a,b)	a if $a>b$, otherwise b
$\min_i(x_i)$	minimum of a sequence of numbers $\{x_i\}$ enumerated by the index i
min(a,b)	a if $a<b$, otherwise b

5 Functional concepts

5.1 Sample grid, sampling and components

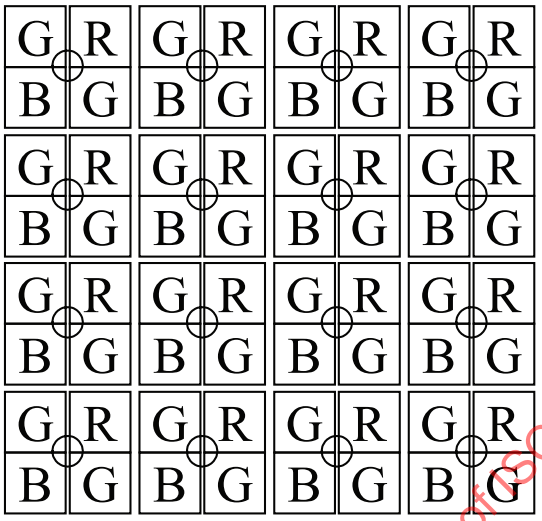
An image is defined as a rectangular array of scalar or vectorial samples regularly aligned along a sample grid of W_f sample positions horizontally and H_f sample positions vertically. W_f is called the *width* and H_f is called the *height* of the image. The vector dimension of the image samples corresponds to the number of colour components present and is indicated by N_c, the number of *components* of the image. Each dimension of this vectorial data corresponds to one *component* of the image, and typically represents one of multiple colour channels of the data. Components may be red, green and blue, or Luma (Y) and Chroma (Cb,Cr). These are only non-exhaustive examples of components, and other uses are possible.

A given component may or may not populate every point on the sample grid. The distance, or *sampling factor*, between sample points of a component shall be constant in each spatial dimension throughout the image. The horizontal and vertical sampling factors of component i of an image are denoted by $s_x[i]$ respectively $s_y[i]$ where i enumerates the components. Annex B provides further specifications on component sampling.

This document *does not* specify how to interpret the sample values, or how to reconstruct from subsampled components an array of samples that populates the entire sample grid, i.e. it does not specify how to *upsample* components to the full resolution of the sampling grid. The sampling grid provides only an abstract coordinate system for the computation of positions and dimensions of codestream elements.

5.2 Interpretation of CFA data

This document defines coding tools and signalling for compression of Bayer-type CFA image data. According to this specification, each sampling grid point represents a super pixel of four sensor elements containing at least one sample of each component. Thus CFA data is interpreted as an image having four components, where each sampling grid point describes four spatially disjoint sensor elements (one element per channel).



Squares represent individual sensor elements and circles represent sampling grid points. Groups of four sensor elements overlapping with the same sampling grid point form one super pixel.

Figure 1 — Example of the interpretation of a GRBG Bayer-type CFA image

Moreover, regardless of the CFA sensor spatial subpixel arrangement, the Star-Tetrix colour transform of this document defines a strict order on the components assigning the red channel to component 0, the green channels to components 1 and 2, and the blue channel to component 3. The spatial subpixel arrangement is signalled by the CRG marker. Figure 1 shows only one of the four potential subpixel arrangements of a Bayer-type CFA.

5.3 Wavelet decomposition

This document provides an efficient representation of image signals through the mathematical tool of wavelet analysis. The wavelet filter process specified in Annex E separates each component into multiple *bands*, where each band consists of multiple *coefficients* describing the image signal of a given component within a frequency domain specific to the *wavelet filter type*, i.e. the particular filter corresponding to the band.

Wavelet coefficients are grouped into *precincts*, where each precinct includes all coefficients over all bands that contribute to a spatial region of the image. Each precinct is encoded into one or multiple *packets* in the codestream syntax specified in Annex A.

Precincts are furthermore grouped into *slices*. Wavelet coefficients in precincts that are part of different slices can be decoded independently from each other. Note, however, that the wavelet transformation runs across slice boundaries. A slice always extends over the full width of the image, but may only cover parts of its height. Bands, band types, precincts and slices are formally defined in Annex B.

5.4 Codestream

The codestream is a linear stream of bits from the first bit to the last bit. For convenience, it can be divided into (8-bit) bytes, starting with the first bit of the codestream. Bits within bytes are enumerated from the LSB to the MSB, with the least significant bit having the index zero.

Annex A specifies the codestream syntax that defines the coded representation of compressed image data for exchange between application environments. Any compressed image data shall comply with the syntax and code assignments appropriate for the decoding processes defined in this document.

The codestream consists of multiple syntax elements: *marker segments* define control information necessary to steer the decoding process, and *entropy coded data* organized in *packets* that represent image information itself. Packets are further grouped into *subpackets*, each of which includes particular information such as magnitude, signs or significance of parts of the encoded image data.

All marker segments defined in this text are specified in Annex A. This annex also provides an overview on the organization of the codestream. Packets and subpackets are specified in Annex C.

6 Encoder requirements

An encoder is an embodiment of a process that generates a codestream that conforms to the syntactical requirements specified in Annex A. Annex C to Annex G include informative subclauses that indicate how an encoder may be implemented.

7 Decoder

7.1 Decoding process general provisions

Figure 2 provides an overview on the decoding process and the layout of this document. Codestream decoding can be grouped into a syntax analysis part in block 1, an entropy decoding stage consisting of multiple blocks 2.1 to 2.4, an inverse quantization in block 3, an inverse wavelet transformation in block 4 and an inverse multiple component transformation in block 5. In block 6, sample values are scaled, a DC offset is added, and they are clamped to their nominal ranges.

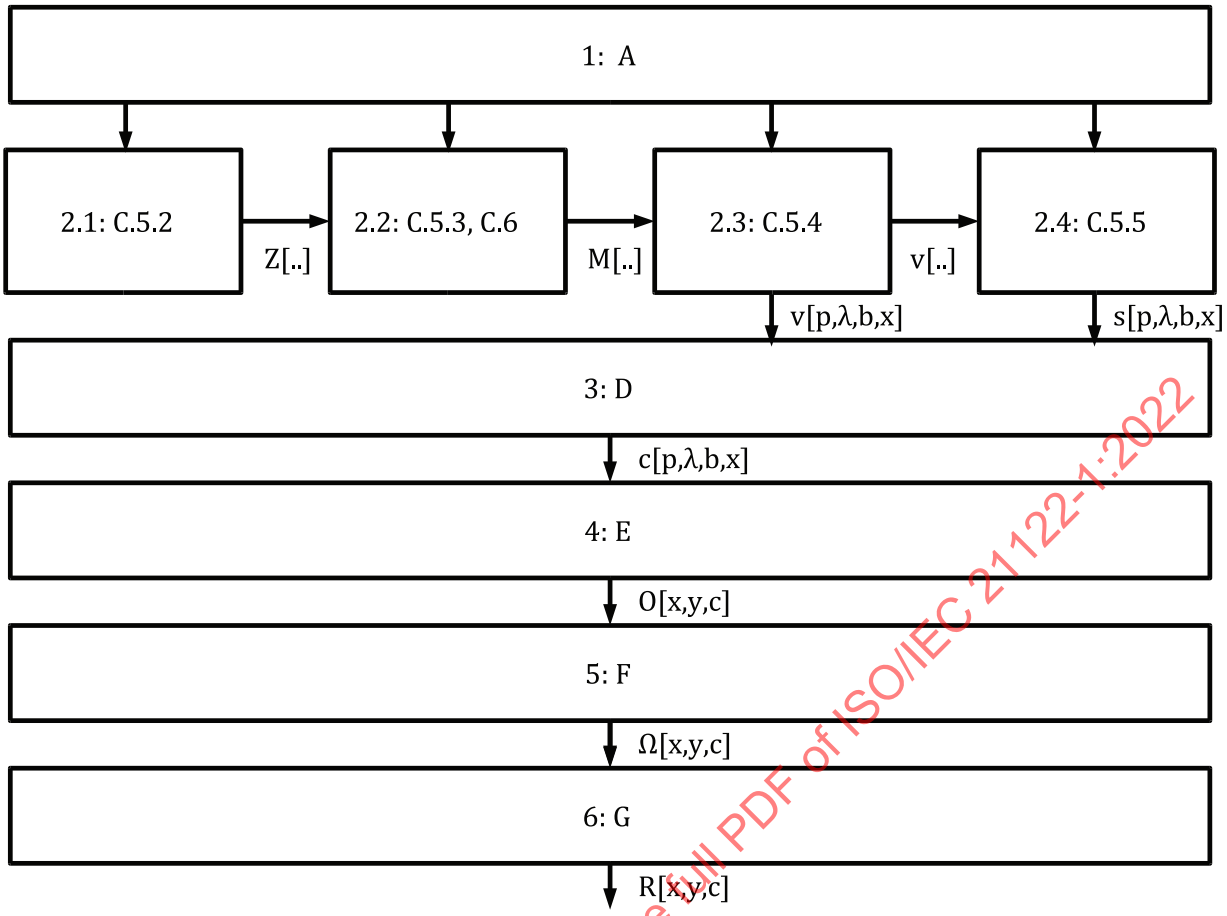


Figure 2 — Decoder overview

In Block 1, described in Annex A, the decoder analyses the codestream syntax and retrieves information on the layout of the sampling grid, and the dimensions of slices and precincts.

The subpackets of the entropy coded data segment of the codestream are then decoded by the procedures given in Annex C to form significance information, sign information, bitplane count information and quantization indices. This operation is performed in blocks 2.1 to 2.4 in Figure 2.

In block 2.1, significance information is decoded from the *significance subpacket* as specified in subclause C.5.2. Denoted by the array $Z[p,\lambda,b,j]$, significance information indicates the presence of significant code groups within the j^{th} significance group. Each significance group corresponds to a run of code groups indexed by precinct p, line λ and band b. A code group is significant if, depending on the Run Mode flag *Rm* in the picture header, it either contains non-zero coefficients, or has a non-zero bitplane count prediction residual.

In block 2.2, bitplane counts are decoded from the *Bitplane count subpacket* as specified in subclause C.5.3 by the procedures specified in subclause E.6 The integer array $M[p,\lambda,b,g]$ indicates the bitplane counts of the wavelet coefficients in the code group g indexed by precinct p, line λ and band b.

In block 2.3, Quantization index magnitudes $v[p,\lambda,b,x]$ in precinct p, line λ, band b, and horizontal position x are decoded from the *data subpacket* as specified in subclause C.5.4.

In block 2.4, the signs of the quantization indices $s[p,\lambda,b,x]$ are either interleaved in the data subpacket, or included in a separate sign subpacket as specified in subclause C.5.5.

In block 3, decoded quantization index magnitudes $v[p,\lambda,b,x]$ and signs $s[p,\lambda,b,x]$ are then inversely quantized by the dequantizer specified in Annex D, giving wavelet coefficients $c[p,\lambda,b,x]$.

In block 4, wavelet coefficients $c[p,\lambda,b,x]$ are inversely wavelet transformed by the procedure specified in Annex E. This process generates spatial sample values for all components, denoted by $O[x,y,c]$. Coordinates x and y are here subsampled sampling grid positions of component c.

In block 5, spatial sample values $O[x,y,c]$ undergo optionally an inverse multiple component transformation, giving intermediate image sample values $\Omega[x,y,c]$. The inverse multiple component transformation is specified in Annex F.

In block 6, a DC offset is added to the decorrelated sample values $\Omega[x,y,c]$, an optional non-linear transformation is applied, they are scaled to their nominal range and then clamped to the range of the bit-precision of the output, giving the final reconstructed output sample values $R[x,y,c]$ populating the sample grid positions $x \times s_x[c], y \times s_y[c]$. This procedure is specified in Annex G.

7.2 Decoder requirements

A decoder is an embodiment of the decoding process. The decoding process converts a codestream by performing the process specified in this document to sample values arranged on a rectangular sampling grid. Annexes A to G describe and specify the decoding process. All decoding processes are normative. Decoder conformance and test procedures to test for conformance are specified in ISO/IEC 21122-4.

There is no normative or required specification for the particular internal steps or ordering of internal operations to be performed within the decoder that is used to produce the normatively specified result. Only the result that is externally observable as the decoded output image produced by the decoder is required to match the result produced by the decoding processes specified in this document up to a conformance-level dependent error bound that is specified in ISO/IEC 21122-4. The descriptions use particular implementation techniques for illustrative purposes only, and any implementation that is able to reproduce the same result as those generated by the algorithms specified herein is conforming to this document.

Annex A
(normative)

Codestream syntax

A.1 General

A.1.1 Marker segments and entropy coded data

The compressed data format consists of an ordered collection of syntax elements. This document distinguishes between three types of syntax elements: *Marker*, *marker segments* and *entropy coded data*. Markers serve to identify the various structural parts of the codestream. Most markers start marker segments, where marker segments signal the characteristics of the encoded image and encapsulate parameters configuring the decoder. Some markers stand alone. Entropy coded data consists of the input to the decoding procedure described in Annexes C to G which reconstructs this data to the output image.

A.1.2 Key to syntax information

JPEG XS codestream syntax elements belong to one of two categories: *fixed-length* numerical values, or *variable-length* codes. In the syntax tables, the "Syntax" column indicates the category to which each codestream syntax element belongs, in the "Size" column the size of each field is identified (if applicable). Fixed-length numerical values and are unsigned integers and are denoted by `u(n)`, where *n* is the number of bits used to represent the value. Variable-length codes are denoted by `vlc`, see subclause C.7 for the normative decoding procedure of variable length codes. Bit strings and variable-length codes appear in the codestream with the left bit first; numerical values appear most-significant bit first. The notation `pad(n)` indicates a variable number padding bits. Padding aligns the bitstream to an *n*-bit boundary, i.e. to an integer multiple of *n* bits relative to the start of the bitstream. Thus, `pad(n)` expands to 0 to *n*−1 bits depending on the position within the bitstream. While padding bits can have arbitrary values, a decoder shall ignore their value. The notation `fill()` indicates an arbitrary number of filler bytes a decoder shall remove without interpreting their value. The amount of filler bytes can be inferred from a length field of a corresponding syntax element.

Syntax elements may be conditionally included in the codestream; this is indicated by `if` clauses in the Syntax column of the syntax tables. All syntactical elements enclosed in curly brackets following the `if` clause are only included if the expression following the `if` clause is non-zero.

The sequence of multiple similar elements is indicated by `for` clauses in the Syntax column of the syntax tables. The elements to repeat are enclosed in curly braces. The loop itself is specified through three syntax elements: an initializer setting a dummy count variable indicating the current iteration position of the loop, a condition on the count variable for continuing the loop, and an iteration statement that updates the count variable for the next loop. The three expressions are separated by semicola.

NOTE The loop syntax and the syntax for conditional inclusion of elements follow closely the syntax of the C language.

A.2 Codestream syntax general provisions

A JPEG XS codestream describes an *image* consisting of 1 to 8 *components* aligned along a regular rectangular sampling grid. Each component is a rectangular arrangement of integer sample values on the sampling grid of the image. The samples of a component need not populate every possible position on the sampling grid, see subclause B.1. The horizontal and vertical spacing between populated sample positions of a component relative to the sampling grid are denoted the *horizontal* and *vertical sampling*

factors of the component and are indicated by the symbols $s_x[i]$ and $s_y[i]$. Subsampling factors vary between 1 and 2. The codestream reconstruction process described by this document assigns to each sample of each image component an integer precision between 8 and 16 bits.

The dimensions of the image, along with the number of components, the sampling factors and the precision of the components are encoded in a syntax element denoted as *Picture Header.* The samples in the image are reconstructed by an inverse wavelet transformation from entropy coded wavelet coefficients arranged in multiple wavelet bands, see Annex B for details. The wavelet reconstruction algorithm is specified in Annex E.

Wavelet coefficients are grouped into precincts. A *precinct* includes entropy coded data decoding to a rectangular array of wavelet coefficients per bands such that the included wavelet coefficients contribute to a given spatial region of the image. A precinct is represented in the codestream by a *precinct header*, *entropy coded data* followed by *filler bytes*.

The entropy coded data is organized in multiple *packets*, where each packet *s* contributes to one line λ and one or multiple bands *b* of a precinct *p*. Each packet consists of multiple subpackets where each subpacket contributes to one aspect of the data, such as *significance, bitplane counts, magnitude* and *signs.* Filler bytes and padding does not have any impact on the decoded image. The purpose of the filler bytes is to prevent buffer underflow of a potential transmission buffer, the purpose of padding is to align the syntax elements in the bitstream to byte boundaries. Decoders learn the number of filler bytes following the precinct from the precinct header specified in subclause C.2, and the number of filler bytes in the subpackets from the subpacket header in specified in subclause C.3. Decoders shall ignore the value of the filler bytes.

NOTE 1 Filler bytes are in no relation to the output of the `pad()` function defined in subclause A.1.2 whose purpose is to ensure alignment of the codestream to byte boundaries only.

NOTE 2 Buffer models, profiles and levels are specified in ISO/IEC 21122-2 and are beyond the scope of this document.

Precincts are grouped into *slices*. Each slice consists of an integral number of precincts, and extends over the full width of the image. Even though the wavelet coefficients within each slice can be decoded independently, the wavelet transformation runs across slices. A slice is represented in the codestream by a *slice header* and one or multiple *precincts* following the slice header.

Figure 2 gives an overview of the hierarchy of JPEG XS codestream syntax structures, Table A.1 defines the overall codestream syntax

Table A.1 — JPEG XS codestream syntax overview

Syntax	Notes	Defined in
`Picture() {`		
`SOC_marker()`	Identifies this codestream as JPEG XS codestream	Table A.3
`capabilities_marker()`	Identifies the capabilities a decoder needs to support to be able to decode the codestream	Table A.6
`picture_header()`	Defines the overall structure of the codestream	Table A.7
`component_table()`	Defines the precision and sampling factors of all components in the image	Table A.15
`weights_table()`	Defines weight and gain factors that steer the decoding process.	Table A.24
`nonlinearity_marker()`	Optional definition of non-linearities for component reconstruction	Table A.16
`cwd_marker()`	Optionally disable wavelet decomposition on some components	Table A.18

Table A.1 *(continued)*

Syntax	Notes	Defined in
`cts_marker()`	Colour transformation specification for the star-tetrix transformation	Table A.19
`crg_marker()`	Optional component registration, mandatory if the star-tetrix transformation is used	Table A.21
`extension_marker()`	Optional extension of the codestream syntax	Table A.22
`for(t=0,p=0;!endofimage;t=t+1) {`	Loop over all slices until all wavelet coefficients of the image have been decoded, where *t* is the slice index and *p* the precinct index	
`slice_header()`	Identifies the ordering of slices	Table A.25
`for(u=0;u<`N_p`[t];p=p+1,u=u+1) {`	Loops over all precincts in a slice, where *t* is the slice index, *p* is the precinct index and *u* enumerates precincts within a slice	
`compute_packet_inclusion(p)`	Determine the packets that are part of this precinct.	Table B.4
`precinct_header(p)`	Defines prediction modes and the quantization of the precinct	Table C.1
`for(s=0;s<`N_{pc}`[p];s=s+1) {`	Loop over all packets of this precinct	
`packet_header(p,s)`	Defines flags and sizes of the packet	Table C.3
`packet_body(p,s)`	Contains the entropy coded data of this packet	Table C.3
`}`	End of loop over subpackets	
`fill()`	Possible byte-aligned filler bytes at the end of the precinct to reach the target bitrate. A decoder shall ignore this data. Subclause C.2 specifies how to determine the number of filler bytes.	Subclause C.2
`}`	End of loop over precincts within a slice	
`}`	End of loop over slices	
`EOC_marker()`	Identifies the end of the JPEG XS codestream	Table A.4
`}`		

NOTE 3 The number of packets $N_{pc}[p]$ depends on the precinct index *p* and is computed by the `compute_packet_inclusion(p)` function specified in Table B.4; in particular, the last precinct of the picture will, by this procedure, include less bands than all other precincts if the picture height is not divisible by the precinct height.

A.3 Markers and marker segments

Markers serve the purpose to identify the various structural parts of the codestream format. Markers may either stand alone, or may start *marker segments* containing a related group of parameters. A *marker segment* consists of a marker, followed by a two-byte length field, followed by the parameters in the marker segment, denoted as *payload data* in the following. The two-byte length field identifies the length of the marker segment, which consists of the length of the payload data in bytes, and the size of the length field itself (two bytes). The length field does not include the size of the marker. Parameters are encoded with the most significant byte first, a convention often denoted as *big endian.*

All markers are assigned two-byte codes: a 0xff byte followed by a byte that is not equal to 0x00 or 0xff. Table A.2 lists all markers used by this document, and, in combination with referenced tables from Table A.1, specifies whether they stand alone or introduce a marker segment. The semantics of each marker and associated marker segment are further specified in subclause A.4.

Some marker segments are currently reserved for future ISO/IEC use. Marker segments can be mandatory, optional, or mandatory only if indicated by a capability signalled in the capability marker

segment. Optional marker segments may be ignored by decoder implementations by skipping over the marker segment by using its length field. The capability marker may indicate the marker segments required to correctly decode a given codestream. If a decoder encounters a capability it does not implement, it should abort decoding.

If the length field of a marker segment does not match the specified value or is not in the specified range, the codestream is ill-formed. For that, decoders should check whether the marker length field has its specified value, or is within the range specified in this document.

NOTE 1 It is possible that future editions of this document will include additional fields in the marker segments and hence require an increase in the size the marker segments. Such additional fields can be necessary to decode the codestream. The above ensures that decoders fail properly when attempting to decode an extended codestream syntax that they do not support.

NOTE 2 Other International Standards implement bit-stuffing or byte-stuffing procedures to ensure that markers can be identified uniquely without decoding or interpreting entropy coded data segments. This is **not** the case for this document. If a decoder loses synchronization with the codestream, lower level transport mechanisms are required to regain synchronization as it is not possible to search the codestream for markers; bit patterns within the entropy coded data segment can replicate the byte sequences used to identify marker segments.

Table A.2 — JPEG XS codestream markers

Code assignment	Symbol	Description	Mandatory/Optional	Reference
0xff10	SOC	Start of codestream	Mandatory	A.4.1
0xff11	EOC	End of codestream	Mandatory	A.4.2
0xff12	PIH	Picture header	Mandatory	A.4.4
0xff13	CDT	Component table	Mandatory	A.4.5
0xff14	WGT	Weights table	Mandatory	A.4.11
0xff15	COM	Extension marker	Optional	A.4.10
0xff16	NLT	Nonlinearity marker	Optional	A.4.6
0xff17	CWD	Component-dependent wavelet decomposition marker	Optional	A.4.7
0xff18	CTS	Colour transformation specification marker	Mandatory if *Cpih*=3, shall not be present otherwise.	A.4.8
0xff19	CRG	Component registration marker	Optional, mandatory if *Cpih*=3	A.4.9
0xff20	SLH	Slice header	Mandatory	A.4.12
0xff50	CAP	Capabilities Marker	Mandatory	A.4.3
All other values			Optional	Reserved for future ISO/IEC purposes

A.4 Syntax description of marker segments

A.4.1 Start of codestream

Function: Identifies the codestream as containing an image represented in accordance with this document.

Usage: Shall be the first marker segment in a codestream. There shall be only one SOC marker at the beginning of each JPEG XS codestream.

Table A.3 — Start of codestream marker syntax

Syntax	Notes	Size	Values
`start_of_codestream() {`			
SOC		u(16)	0xff10
}			

A.4.2 End of codestream

Function: Identifies the end of a JPEG XS codestream.

Usage: Shall be the last marker segment in a codestream. There shall be exactly one EOC marker at the end of each JPEG XS codestream.

Table A.4 — End of codestream marker syntax

Syntax	Notes	Size	Values
`end_of_codestream() {`			
EOC		u(16)	0xff11
}			

A.4.3 Capabilities marker

Function: Identifies capabilities required to decode a JPEG XS codestream.

Usage: Shall be the second marker segment in a codestream. There shall be exactly one CAP marker which shall be placed behind the SOC marker.

NOTE 1 The above implies that the CAP marker is always present, even when the *cap*[] array is empty.

Table A.5 specifies the assignment of capabilities to bits, Table A.6 the syntax of the CAP marker segment. Furthermore, *Lcap* shall be selected such that for *Lcap*>2, *cap*[(*Lcap*−2) ×8−8] to *cap*[(*Lcap*−2) ×8−1] are not all 0.

NOTE 2 This condition on the cap array can always be arranged by selecting a smaller Lcap.

NOTE 3 Bit 0 of the capability marker is intentionally unused.

Table A.5 — Capability bit assignment

Bit number *i* in *cap*[] array	Bit value	Meaning
1	0	Support for Star-Tetrix transform not required, and support for CTS marker not required
1	1	Support for Star-Tetrix transform and CTS marker required
2	0	Support for quadratic non-linear transform not required
2	1	Support for quadratic non-linear transform required
3	0	Support for extended non-linear transform not required
3	1	Support for extended non-linear transform required
4	0	$s_y[i]$=1 for all components i
4	1	component i with $s_y[i]$>1 present

Table A.5 *(continued)*

Bit number *i* in *cap*[] array	Bit value	Meaning
5	0	Support for component-dependent wavelet decomposition not required
5	1	Support for component-dependent wavelet decomposition required
6	0	Support for lossless decoding not required
6	1	Support for lossless decoding required
8	0	Support for packet-based raw-mode switch not required
8	1	Support for packet-based raw-mode switch required

Table A.6 — Capabilities marker syntax

Syntax	Notes	Size	Values
`capabilities_marker() {`			
`CAP`		u(16)	0xff50
`Lcap`	Size of the capabilities marker in bytes (not including the marker)	u(16)	Variable
`for(i=0;i<(Lcap-2)×8;i=i+1) {`	Loop over capabilities bits		
`cap[i]`	Requirement of capability i. *cap*[*i*] is 1 if capability *i* is required for decoding a codestream, and 0 otherwise.	u(1)	
`}`	End of loop over capabilities bits		
`padding`	Pad to an integer number of bytes	pad(8)	
`}`			

A.4.4 Picture header

Table A.7 defines the syntax of the picture header.

Function: Provides information on the dimensions of the image, the precision of its component and the configuration of the decoder.

Usage: Shall be the third marker segment in a codestream directly after the CAP marker. There shall be exactly one PIH marker in a JPEG XS codestream.

Table A.7 — Picture header syntax

Syntax	Notes	Size	Values
`picture_header() {`			
PIH		u(16)	0xff12
Lpih	Size of the segment in bytes (not including the marker)	u(16)	26
Lcod	Size of the entire codestream in bytes from SOC to EOC, including all markers, if constant bitrate coding is used. 0 if variable bitrate coding is used.	u(32)	$0 — (2^{32}-1)$
Ppih	Profile this codestream complies to. Decoders should abort decoding if they identify a profile they do not implement.	u(16)	0 for no restrictions, see ISO/IEC 21122-2 for profiles.
Plev	Level and sublevel to which this codestream complies. Decoders should abort decoding if they identify a level they do not support.	u(16)	0 for no restrictions, see ISO/IEC 21222-2 for additional levels.
$\mathbf{W_f}$	Width of the image in sample grid positions	u(16)	$\max_i(s_x[i])\times 2^{N_{L,x}} — (2^{16}-1)$
$\mathbf{H_f}$	Height of the image in sample grid positions	u(16)	$\max_i(s_y[i])\times 2^{N_{L,y}} — (2^{16}-1)$
Cw	Width of a precinct in multiples of $8\times\max_i(s_x[i])\times 2^{N_{L,x}}$ sample positions, other than the rightmost precincts. If this field is 0, precincts are as wide as the image. See subclause B.5 for details.	u(16)	$0 — (2^{16}-1)$ such that either $Cw=0$ or $W_f \bmod (8\times C_w\times\max_i(s_x[i])\times 2^{N_{L,x}}) \geq \max_i(s_x[i])\times 2^{N_{L,x}}$
$\mathbf{H_{sl}}$	Height of a slice in precincts other than the last slice	u(16)	$1 — (2^{16}-1)$
$\mathbf{N_c}$	Number of components in the image	u(8)	1—8
$\mathbf{N_g}$	Number of coefficients per code group	u(8)	4
$\mathbf{S_s}$	Number of code groups per significance group	u(8)	8
Bw	Nominal bit precision of the wavelet coefficients	u(8)	20,18 or $B[0]$ [a]
Fq	Number of fractional bits in the wavelet coefficients	u(4)	8,6 or 0 [a]
$\mathbf{B_r}$	Number of bits to encode a bitplane count in raw	u(4)	4
Fslc	Slice coding mode	u(1)	See Table A.14
Ppoc	Progression order of bands within precincts	u(3)	See Table A.13
Cpih	Colour transformation to be used for inverse decorrelation	u(4)	See Table A.9
$\mathbf{N_{L,x}}$	Number of horizontal wavelet transformations	u(4)	1–8
$\mathbf{N_{L,y}}$	Number of vertical wavelet transformations of non-vertically subsampled components	u(4)	$\max_i(\log_2(s_y[i]))-\min(N_{L,x},6)$
Lh	Long header enforcement flag	u(1)	0 or 1
Rl	Raw-mode selection per packet flag	u(1)	0 or 1
Qpih	Inverse quantizer type	u(2)	See Table A.10
Fs	Sign handling strategy	u(2)	See Table A.11
Rm	Run mode	u(2)	See Table A.12

[a] See Table A.8 for valid combinations of *Bw* and *Fq* and further restrictions.

NOTE 1 The condition on *Cw* ensures that all but the rightmost precincts have in the LL band at least 8 samples, and that all bands of the rightmost precincts are non-empty.

NOTE 2 In case $Wf \times N_c > 16376$ and $N_{L,y}=0$, or for components that do participate in the wavelet decomposition, it can happen that $L_{sgn}[p,s] \geq 2048$ or $L_{dat}[p,s] \geq 32768$. As such sizes cannot be expressed by the short header, the *Lh* flag allows to enforce long headers. *Lh* can safely remain 0 as long as at least one vertical decomposition is performed.

Table A.8 — Valid combinations of *Bw* and *Fq*

Bw	*Fq*	Additional constraints	Notes
$B[0]$	0	$B[i]=B[0]$ for all i.	This combination indicates lossless coding.
18	6	A NLT marker segment shall be present. Bit 2 or bit 3 of the CAP marker segment shall be 1, see subclause A.4.3	This combination is used in case a non-linearity is present.
20	8	Bits 2 and 3 of the CAP marker segment not present or shall be 0.	Regular case.

Table A.9 — Colour transformation

Cpih	Meaning
0	No colour transform
1	Reversible RGB to YCbCr colour transformation (see Annex F)
3	Star-Tetrix transform (see Annex F)
2,4–15	Reserved for ISO/IEC purposes

Table A.10 — Quantizer type

Qpih	Meaning
0	Deadzone quantizer (see Annex D)
1	Uniform quantizer (see Annex D)
2–3	Reserved for ISO/IEC purposes

Table A.11 — Sign handling strategy

Fs	Meaning
0	Signs encoded jointly with the data
1	Signs encoded separately
2–3	Reserved for ISO/IEC purposes

Table A.12 — Run mode

Rm	Meaning
0	Runs indicate zero prediction residuals
1	Runs indicate zero coefficients
2–3	Reserved for ISO/IEC purposes

Table A.13 — Progression order

Ppoc	Meaning
0	The progression order as defined by subclause B.7 (resolution-line-band-component)
1–7	Reserved for ISO/IEC purposes

Table A.14 — Slice coding mode

Fslc	Meaning
0	The wavelet transformation runs across slice boundaries
1	Reserved for ISO/IEC purposes

A.4.5 Component table

Table A.15 defines the syntax of the component table.

Function: This marker segment specifies the component precision and the sampling factors of each component in the image. The number of components itself is given by the N_c parameter of the picture header.

Usage: There shall be exactly one component table in each JPEG XS codestream. It shall precede the first slice header.

Table A.15 — Component table syntax

Syntax	Notes	Size	Values
`component_table() {`			
`CDT`		u(16)	0xff13
`Lcdt`	Size of the segment in bytes, not including the marker	u(16)	$2 \times N_c + 2$
`for(c=0;c<Nc;c = c + 1) {`	Loop over components. The number of components is specified in the picture header.		
`B[c]`	Bit precision of component *c*	u(8)	8–16
s_x`[c]`	Horizontal sampling factor of component *c*	u(4)	1 or 2 for components 1 and 2, 1 for all other components
s_y`[c]`	Vertical sampling factor of component *c*	u(4)	$1 - s_x[c]$
`}`	End of loop over components		
`}`			

A.4.6 Nonlinearity marker

Table A.16 defines the syntax of the nonlinearity marker segment.

Function: Defines an optional non-linear transform to be applied after inverse multiple component transformation.

Usage: Zero or one nonlinearity marker segments may be present in a codestream. If present, any nonlinearity marker segment shall precede the first slice header in the codestream and shall follow the picture header. This marker shall not be present if **Fq**=0, i.e. in the lossless mode.

Table A.16 — Nonlinearity marker segment syntax

Syntax	Notes	Size	Values
`nonlinearity_marker(){`			
`NLT`		u(16)	0xff16
`Lnlt`	Size of the marker segment, not including the marker	u(16)	5 or 12
`Tnlt`	Type of the non-linearity	u(8)	See Table A.17
`if (Tnlt == 1) {`	Additional data for quadratic non-linearity		

Table A.16 *(continued)*

Syntax	Notes	Size	Values
σ	Read sign bit of the DC offset	u(1)	0—1
α	Read the remaining bits of the DC offset	u(15)	0—2^{15-1}
DCO=α−σ ×2^{15}	Compute the *DCO* offset from two's complement representation.		
}			
if (Tnlt == 2) {	Additional data for extended non-linearity		
T1	Upper threshold for region 1, in units of *Bw*	u(32)	1—2^{Bw}−1
T2	Upper threshold for region 2, in units of *Bw*	u(32)	1—2^{Bw}−1
E	Exponent of the linear slope in region 2	u(8)	1—4
}			
}			

Table A.17 — Tnlt encoding

Tnlt	Meaning
1	Quadratic non-linearity
2	Extended non-linearity
All other values	Reserved for ISO/IEC use

A.4.7 Component-dependent wavelet decomposition marker

Table A.18 defines the syntax of the component-dependent decomposition marker.

Function: Optionally suppresses the wavelet decomposition for one or more components. If this marker is not present, the value of *Sd* shall be 0.

Usage: Zero or one component-dependent wavelet decomposition marker segments may be present in a codestream. Shall only be present if N_c>3. If present, shall precede the first slice header in the codestream and shall follow the picture header.

Table A.18 — Component-dependent wavelet decomposition marker segment syntax

Syntax	Notes	Size	Values
cwd_marker(){			
CWD		u(16)	0xff17
Lcwd	Size of the marker segment, not including the marker	u(16)	3
Sd	Number of components for which the wavelet decomposition is suppressed.	u(8)	1—N_c−1 Furthermore, Sd shall be selected such that all components that are excluded from the wavelet transformation have $s_x[c]$=1 and $s_y[c]$=1
}			

A.4.8 Colour transformation specification marker

Table A.19 defines the syntax of the colour transformation specification marker segment.

Function: Defines parameters of the Star-Tetrix transformation.

Usage: Zero or one colour transformation specification markers shall be present in a codestream. Shall be present if and only if *Cpih*=3, i.e. if the Star-Tetrix transformation is in use. Shall precede the first slice header in the codestream and shall follow the picture header.

Table A.19 — Colour transformation specification marker segment syntax

Syntax	Notes	Size	Values
cts_marker(){			
CTS		u(16)	0xff18
Lcts	Size of the marker segment, not including the marker	u(16)	4
Reserved	Reserved for ISO/IEC purposes	u(4)	0
C_f	Size and extent of the transformation	u(4)	See Table A.20
e_1	Exponent of first chroma component	u(4)	0..3
e_2	Exponent of second chroma component	u(4)	0..3
}			

Table A.20 — C_f encoding

C_f	Meaning
0	Full transformation, access to the line below and above required
3	Restricted in-line transformation, no access to neighbouring lines
All other values	Reserved for ISO/IEC use

A.4.9 Component registration marker

Table A.21 defines the syntax of the component registration marker segment.

Function: Defines the relative placement of the component to the sample grid. If not present, the components are placed at the vertices of the sampling grid.

Usage: Zero or one component registration marker shall be present in a codestream. If *Cpih*=3, exactly one CRG marker segment shall be present. If present, any CRG marker segment shall precede the first slice header in the codestream and shall follow the picture header.

Table A.21 — Component registration marker segment syntax

Syntax	Notes	Size	Values
crg_marker(){			
CRG		u(16)	0xff19
Lcrg	Size of the marker segment, not including the marker	u(16)	Variable, at least 6
for(c=0;c<N_c;c=c+1) {	Loop over all components		
Xcrg[c]	Relative horizontal placement of component c to the sample grid points in units of 1/65536. 0 indicates placement at the horizontal position of the sample grid, a positive value a displacement to the right of the sample grid position. A value of 32768 corresponds to a placement midway between the sample grid point and the next sample grid point to the right.	u(16)	0—65535

Table A.21 *(continued)*

Syntax	Notes	Size	Values
Ycrg[c]	Relative vertical placement of component c to the sample grid points in units of 1/65536. 0 indicates placement at the vertical position of the sample grid, a positive value a displacement to the bottom of the sample grid position. A value of 32768 corresponds to a placement mid-way between the sample grid point and the next sample grid point below.	u(16)	0—65535
}	End of loop over all components		
}			

A.4.10 Extension marker

Table A.22 defines the syntax of the extension marker segment.

Function: Extends the codestream by generic or vendor-specific metadata.

Usage: Zero or more extension marker segments may be present in a codestream. If present, any extension marker segment shall precede the first slice header in the codestream.

Table A.22 — Extension marker segment syntax

Syntax	Notes	Size	Values
extension_marker(){			
COM		u(16)	0xff15
Lcom	Size of the marker segment, not including the marker	u(16)	Variable, at least 4
Tcom	Type of the extension	u(16)	See Table A.23
Dcom	User-defined data	Variable	Variable
Padding	Padding to an integer number of bytes	pad(8)	0
}			

Table A.23 — Tcom encoding

Tcom	Meaning
0x0000	Vendor of the encoder, Dcom is a zero-terminated, ISO 10646 encoded string identifying the vendor of the encoder
0x0001	Copyright statement that the codestream creator want to convey to users of the code-stream. The interpretation of this statement is beyond the scope of this document. Dcom is a zero-terminated, ISO/IEC 10646 encoded string identifying this statement.
0x8000-0xffff	Vendor-specific information. Tcom identifies the type of extension and the vendor.
All other values	Reserved for ISO/IEC use

A.4.11 Weights table

Table A.24 defines the syntax of the weights table.

Function: This marker segment contains parameters required to set the gain of each band relative to the precinct quantization. Together with the parameters in the precinct header, see subclause C.2, this allows to determine the quantization of the wavelet coefficients in this band. Details on how to use these parameters are specified in subclause C.6.2. The number of wavelet bands and the relation between band, component and wavelet filter type are specified in Annex B.

Usage: There shall be exactly one weights table in each JPEG XS codestream. This marker shall appear before the first slice header in the codestream and shall follow the picture header.

Table A.24 — Weights table syntax

Syntax	Notes	Size	Values
weights_table() {			
WGT		u(16)	0xff14
Lwgt	Size of the segment in bytes, not including the marker	u(16)	Variable
for(b=0;b < N_L; b = b+1) {	Loop over all bands.		
if (b'$_x$[b]) {	Check whether band *b* exists		
G[b]	Gain of band *b*	u(8)	0–15
P[b]	Priority of band *b*	u(8)	0-255
}	End of test whether *b* exists.		
}	End of loop over bands		
}			

NOTE Example weights tables for various configurations are given in Annex H. These configurations have been optimized for PSNR performance of the encoder. Other choices are possible and can result in improved visual quality for a certain viewing distance.

A.4.12 Slice header

Table A.25 defines the syntax of the slice header.

Function: This marker segment identifies the start of a slice and provides sufficient information for the order of the slices within the image.

Usage: A codestream contains one or more slice headers. The entropy coded data for one slice follows the slice header and extends either to the next slice header or the end of the codestream. Even though the slice header includes the relative order of the slice in the image, a codestream shall contain slices in incremental order, i.e. progressing from the top of the image to the bottom of the image.

Table A.25 — Slice header syntax

Syntax	Notes	Size	Values
slice_header() {			
SLH		u(16)	0xff20
Lslh	Size of the segment in bytes, not including the marker	u(16)	4
Yslh	Index of the slice, counting from line 0 (at the top of the image) downwards (towards the bottom of the image)	u(16)	0–(2^{16}–1)
}			

NOTE Slice indices ease to regain synchronization in case transmission errors corrupted the codestream.

Annex B
(normative)

Image data structures

B.1 Dimensions of chroma subsampled image planes

An image consists of N_c components aligned along a regular rectangular sampling grid. Each component is a rectangular arrangement of integer sample values aligned to the sampling grid of the image. The samples of a component are not required to populate every possible position on the sampling grid. The horizontal spacing between samples of component i is denoted by $s_x[i]$ and is called the *horizontal sampling factor*. The vertical spacing of component i is denoted by $s_y[i]$ and is called the *vertical sampling factor*. Both, horizontal and vertical sampling factors, are signalled in the component table specified in subclause A.4.5. Figure B.1 provides an example of how samples are assigned to positions on the sampling grid.

The number of samples in every row of component i is given by

$$W_c[i] = \left\lceil \frac{W_f}{s_x[i]} \right\rceil$$

The number of samples in each column of component i is given by

$$H_c[i] = \left\lceil \frac{H_f}{s_y[i]} \right\rceil$$

W_f and H_f are the horizontal and vertical dimensions of the sampling grid of the image as signalled in the picture header.

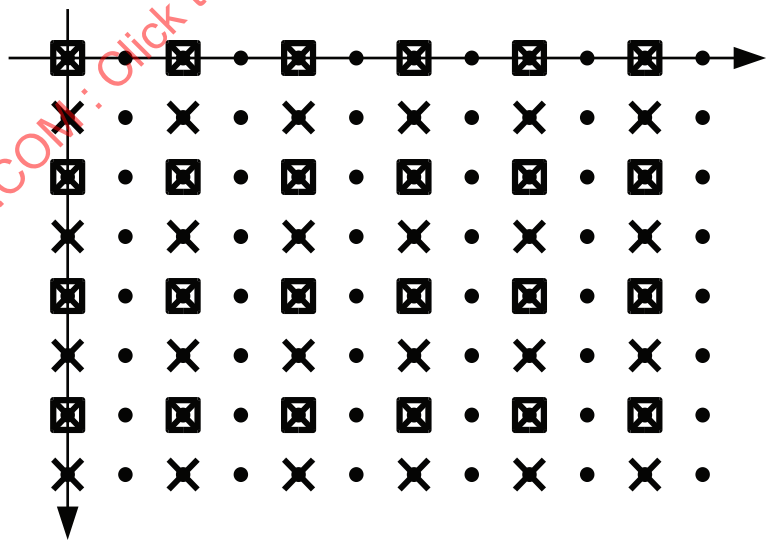


Figure B.1 — Sampling grid for 4:2:2 sampling

NOTE 1 In Figure B.1, $s_x[0]=s_y[0]=1$ is indicated by round dots, $s_x[i]=2$ $s_y[i]=1$ coresponding to 4:2:2 sampling is indicated by crosses, $s_x[i]=2$ $s_y[i]=2$ corresponding to 4:2:0 sampling is indicated by boxes and arrows point towards increasing x and y position. Even though this figure indicates the position of the sample values on the sampling grid from which inclusion or exclusion of sample values in image structures such as bands or precincts is derived, the figure cannot be taken as an indication how a full-scale image should be reconstructed from a 4:2:2 or 4:2:0 sampled image or how components are registered relative to each other; such information is either provided by the CRG marker, if present, or can be derived by means beyond this document. In particular, both centred and co-sited positioning of subsampled components are possible.

NOTE 2 For 4:4:4 sampling, each sample grid point is populated by sample values of all components.

B.2 Division of the subsampled image plane into bands

Each component i is decomposed by a wavelet transformation with $N'_{L,x}[i]$ horizontal and $N'_{L,y}[i]$ vertical decomposition levels, where

$$\begin{matrix} N'_{\mathrm{L,x}}[i] := N_{L,x} \\ N'_{\mathrm{L,y}}[i] := N_{\mathrm{L,y}} - \log_2\left(s_y[i]\right) \end{matrix} \quad \text{for } i < N_c - Sd, \text{ and} \quad \begin{matrix} N'_{\mathrm{L,x}}[i] := 0 \\ N'_{\mathrm{L,y}}[i] := 0 \end{matrix} \quad \text{for } i \geq N_c - Sd$$

That is, the number of horizontal decomposition levels applied to component i is given by the maximal horizontal decomposition level $N_{L,x}$ and the vertical wavelet decomposition depth depends on the vertical subsampling factor $s_y[i]$ and the maximal decomposition level $N_{L,y}$ for all components that participate in the wavelet decomposition and is otherwise 0. For vertically subsampled components, the number of vertical decomposition levels is reduced by 1.

Band types are enumerated by two letters indicating the horizontal and vertical filter type, each of which can be either H indicating high-pass filtering or L for low-pass filtering, where the first letter corresponds to the horizontal filter type and the second letter corresponds to the vertical filter type. The filter type is followed by two subscripts indicating the horizontal decomposition level d_x and the vertical decomposition level d_y. The algorithm of the wavelet transformation itself is specified in Annex E. The filter type, as the collection of horizontal and vertical filtering and horizontal and vertical decomposition depth, is collapsed into a single number β by a procedure given by subclause B.3.

The width $W_b[\beta,i]$ of filter type β at $d_x[\beta,i]$ horizontal decompositions of component i is given by:

$$W_b[\beta,i]=\left\lceil \frac{W_c[i]}{2^{d_x[\beta,i]}} \right\rceil \text{ for a horizontally low-pass filtered band and } W_b[\beta,i]=\left\lfloor \left\lceil \frac{W_c[i]}{2^{d_x[\beta,i]-1}} \right\rceil /2 \right\rfloor \text{ for a}$$

horizontally high-pass filtered band.

The height $H_b[\beta,i]$ of filter type β at $d_y[\beta,i]$ vertical decomposition levels of component i is given by:

$$H_b[\beta,i]=\left\lceil \frac{H_c[i]}{2^{d_y[\beta,i]}} \right\rceil \text{ for a vertically low-pass filtered band and } H_b[\beta,i]=\left\lfloor \left\lceil \frac{H_c[i]}{2^{d_y[\beta,i]-1}} \right\rceil /2 \right\rfloor \text{ for a}$$

vertically high-pass filtered band.

In case of 0 vertical wavelet decompositions, a vertical high-pass does not exist and $H_b[\beta,i]$ is identical to $H_c[i]$ for all filter types β contributing to component i.

B.3 Band indices, horizontal and vertical decomposition levels

Bands are the result of the wavelet decomposition of a component i with a wavelet filter β. The wavelet filter type β is an index in the range of 0 to N_β–1, where N_β enumerates the number of different

wavelet filters. N_β is computed from the number of horizontal decomposition levels $N_{L,x}$ and vertical decomposition levels $N_{L,y}$ as follows:

$$N_\beta = \left(2 \times \min(N_{L,x}, N_{L,y}) + \max\left(N_{L,x}, N_{L,y}\right) + 1\right)$$

The wavelet filter type β is computed from the horizontal decomposition depth d_x and vertical decomposition depth d_y as follows: Set τ_x to 1 if the band is a horizontal high-pass, otherwise set τ_x to 0. Similarly, set τ_y to 1 if the band is a vertical high-pass, otherwise set τ_y to 0. Then for $N_{L,x} \geq N_{L,y}$ compute

$$\beta = \begin{cases} \left(N_{L,x} - d_x\right) + \tau_x & \text{if } d_x > d_y \\ \left(N_{L,x} - N_{L,y}\right) + \tau_x + 2\tau_y + 3 \times \left(N_{L,y} - d_y\right) & \text{otherwise.} \end{cases}$$

NOTE 1 For $N_{L,x} < N_{L,y}$ interchange $N_{L,x}$ with $N_{L,y}$ and d_x with d_y. However, this case needs not to be considered in this document.

NOTE 2 For 5 horizontal and 0 vertical levels, the above formula results in Table B.1, for 5 horizontal and 1 vertical levels, the above formula results in Table B.2; for 5 horizontal and 2 vertical levels, it results in Table B.3. The enumeration of bands in the tables follows the language of subclause B.2. It is important to observe that the assignment of wavelet filter types β does not depend on sampling, i.e. the assignment is identical for 4:4:4, 4:2:2 and 4:2:0 sampling.

NOTE 3 For components $i < N_c - Sd$ and $s_y[i] > 1$, the wavelet filter types $\beta = N_{L,x} + 2 - N_{L,y}$ and $\beta = N_{L,x} + 3 - N_{L,y}$ are not populated and do not carry any data. See also subclause B.4.

NOTE 4 For components $i > N_c - Sd$, only the wavelet filter type $\beta = 0$ is populated, all other filter types do not carry data.

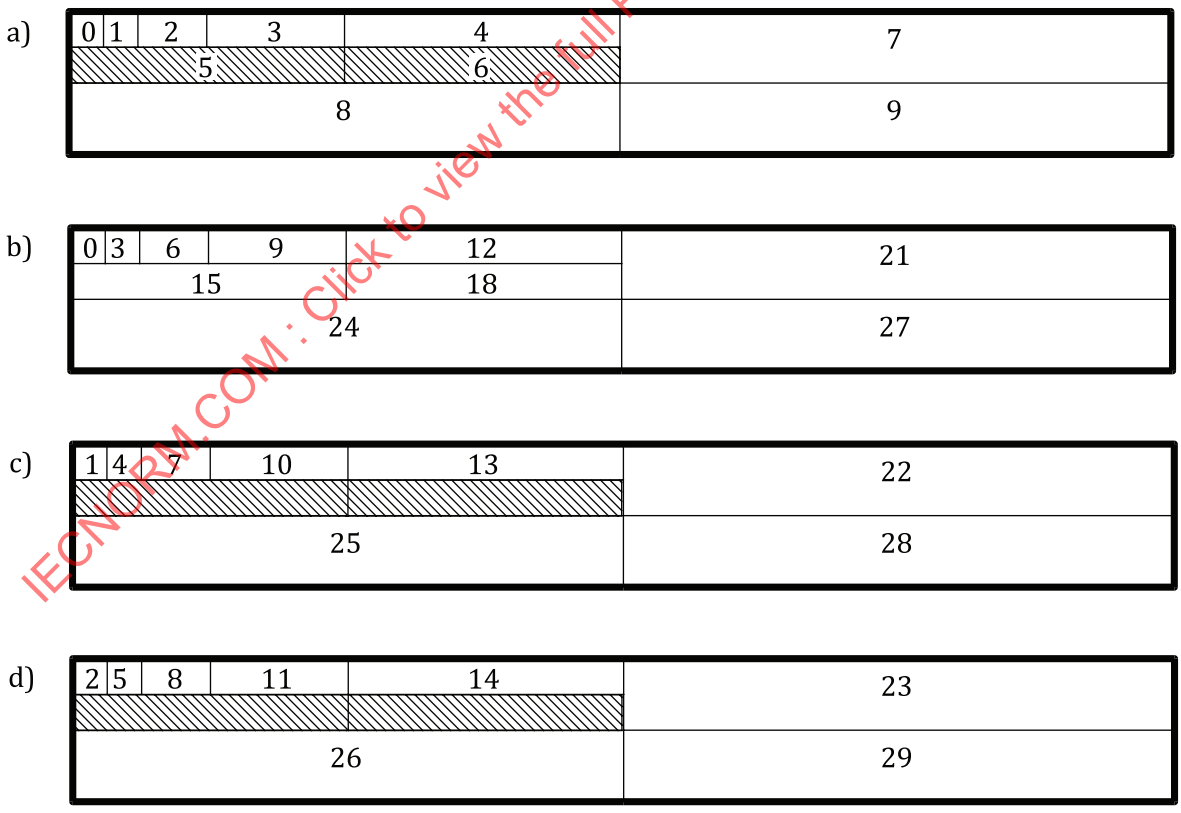


Figure B.2 — Wavelet filter types and band indices for 4:2:0 sampling

NOTE 5 Figure B.2 shows the wavelet filter types in a), the band indices of the luma component in b), the band indices for the Cb component in c) and the band indices for the Cr component in d) for a decomposition with 5 horizontal and 2 vertical levels, 3 components with 4:2:0 sampling and no decomposition suppression.

The functions $d_x[\beta,i]$ and $d_y[\beta,i]$ map a component index i and and a wavelet filter type β into a horizontal or vertical decomposition level.

NOTE 6 The functions $d_x[\beta,i]$ and $d_y[\beta,i]$ are in general component dependent; in the example from Figure B.2, $d_y[0,0]=2$, but $d_y[0,1]=1$.

Bands are enumerated by a single sequential number b. This sequential number is an index in the range 0 to N_L-1, where N_L counts the number of bands. N_L is computed from N_β and the number of components N_c as follows:

$$N_L = (N_c - Sd) \times N_\beta + Sd$$

NOTE 7 If some components are subsampled in the vertical direction or omitted from the wavelet decomposition, N_L deviates from the total number of wavelet bands accumulated over all components. This is intentional. Wavelet filter types that are not present in some components are instead excluded by the mechanism of the band-inclusion flags $I[p,b,\lambda,s]$ and the band-existence flags $b_x[\beta,i]$.

For $i<N_c-Sd$, the band index b is computed from the wavelet filter type β and the component index i in the following way:

$$b = (N_c - Sd) \times \beta + i$$

For $i \geq N_c-Sd$, only $\beta=0$ is populated, all other bands are empty, and the band index b is given by

$$b = (N_c - Sd) \times N_\beta + i$$

NOTE 8 By the above convention, iterating over increasing b corresponds to a progression order with the component index as fast and the wavelet filter type as slow variable. Components that participate in the wavelet decomposition are transmitted in the first block, followed by all components which are not decomposed.

B.4 Band existence flags

In case the wavelet decomposition is suppressed, not all of the N_β wavelet filter types are present for all components, see Figure B.2 for an example. The presence of a band given a filter type and a component is encoded in the array $b_x[\beta,i]$. The value of $b_x[\beta,i]$ is 1 if the wavelet filter type β is present in in component i, and is 0 if it does not exist. The value of this array shall be computed as as follows:

$$b_x[\beta,i] = \begin{cases} 0 & \text{if} \quad \beta > 0 \text{ and } i \geq N_c - Sd \text{ or} \\ 0 & \text{if} \quad 2^{\max(N_{L,y} - d_y[\beta])} \times \tau_y[\beta] \bmod s_y[i] \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

That is, a band is excluded if either the wavelet filter type is nonzero and the component is not decomposed, or the first line of the wavelet band is not divisible by the vertical subsampling factor. Note that the right hand side of the middle condition is identical to $L_0[p,b]$, the first line of the band, defined in subclause B.6. The shaded areas of Figure B.2 corresponds to wavelet filter types β for which $b_x[\beta,1]=0$.

The band-indexed band-existence flag $b'_x[b]$ is 1 if and only if the band b corresponding to filter type β and component i exists, and is 0 otherwise. It can be derived from the band existence flags $b_x[\beta,i]$ as follows:

$$b'_x[(N_c - Sd) \times \beta + i] = b_x[\beta,i] \quad \text{for } i < N_c - Sd$$

$$b'_x\left[(N_c - Sd) \times N_\beta + i\right] = b_x[\beta,i] \quad \text{for } i \geq N_c - Sd$$

Table B.1 — Wavelet filter types for 0 vertical and 5 horizontal decomposition levels

Wavelet filter type β	Wavelet filtering and decomposition depths (subscripts are d_x and d_y)
0	$LL_{5,0}$
1	$HL_{5,0}$
2	$HL_{4,0}$
3	$HL_{3,0}$
4	$HL_{2,0}$
5	$HL_{1,0}$

Table B.2 — Wavelet filter types for 1 vertical and 5 horizontal decomposition levels

Wavelet filter type β	Wavelet filtering and decomposition depths (subscripts are d_x and d_y)
0	$LL_{5,1}$
1	$HL_{5,1}$
2	$HL_{4,1}$
3	$HL_{3,1}$
4	$HL_{2,1}$
5	$HL_{1,1}$
6	$LH_{1,1}$
7	$HH_{1,1}$

Table B.3 — Wavelet filter types for 2 vertical and 5 horizontal decomposition levels

Wavelet filter type β	Wavelet filtering and decomposition depths (subscripts are d_x and d_y)
0	$LL_{5,2}$
1	$HL_{5,2}$
2	$HL_{4,2}$
3	$HL_{3,2}$
4	$HL_{2,2}$
5	$LH_{2,2}$
6	$HH_{2,2}$
7	$HL_{1,1}$
8	$LH_{1,1}$
9	$HH_{1,1}$

NOTE In the above tables, the wavelet filter types are indicated by two capital levels, giving the type of the wavelet filter in horizontal and vertical direction, and two subscripts, counting the number of decompositions that have been applied in horizontal and vertical direction. A letter H indicates high-pass filtering, a letter L low-pass filtering. For all of the above, *Sd*=0.

B.5 Division of the wavelet-transformed image into precincts

The wavelet coefficients are partitioned into a rectangular grid of $N_{p,x} \times N_{p,y}$ precincts, where $N_{p,x}$ is the number of precincts per line of the sampling grid, and $N_{p,y}$ is the number of precincts per column. Each column other than the rightmost column is

$$C_s = \begin{cases} 8 \times Cw \times \max_i(s_x[i]) \times 2^{N_{L,x}} & \text{if } Cw > 0 \\ W_f & \text{otherwise} \end{cases}$$

sample grid positions wide, where W_f and Cw are signalled in the picture header, and the $s_x[i]$ are signalled in the component table. Each precinct contains coefficients from a rectangular area of coefficients from all bands. These coefficients, in turn, correspond to a rectangular region of image data.

NOTE While not a requirement specified in this document, it is generally advisable to select Cw in such a way that the rightmost column is approximately of the same size as that of all other columns.

The number $N_{p,x}$ of precincts per line and the number $N_{p,y}$ of precincts per column are defined as follows:

$$N_{p,x} = \left\lceil \frac{W_f}{C_s} \right\rceil \quad \text{and} \quad N_{p,y} = \left\lceil \frac{H_f}{2^{N_{L,y}}} \right\rceil$$

where W_f is the width of the sampling grid, H_f is the height of the sampling grid, C_s is the column width in sampling grid positions and $N_{L,y}$ the number of vertical decomposition levels. W_f, H_f and $N_{L,y}$ are signalled in the picture header specified in subclause A.4.3.

Precincts are assigned sequential numbers p, denoted as *precinct indices*, where p runs from 0 to $N_{p,x} \times N_{p,y} - 1$ with $N_{p,x}$ and $N_{p,y}$ defined as above. The sequential number enumerates the precincts on the sampling grid in a raster scan manner, i.e. firstly from left to right, and secondly from top to bottom.

Precinct number p is $H_p = 2^{N_{L,y}}$ sampling grid lines high and $W_p[p]$ sampling grid columns wide, where the latter is computed as

$$W_p[p] = \begin{cases} C_s & \text{if } p \bmod N_{p,x} < N_{p,x} - 1 \\ ((W_f - 1) \bmod C_s) + 1 & \text{otherwise} \end{cases}$$

Denote by (x_b, y_b) the coefficient positions within band b, where b is the band corresponding to filter type β and component i as defined in subclause B.3. Then x_b runs from 0 to $W_b[\beta,i]-1$ and y_b runs from 0 to $H_b[\beta,i]-1$, with $W_b[\beta,i]$ and $H_b[\beta,i]$ the band dimensions as defined in subclause B.2. A sample in band b at position (x_b, y_b) is part of precinct p if and only if

$$\left\lfloor \frac{p}{N_{p,x}} \right\rfloor = \left\lfloor \frac{y_b \times s_y[i] \times 2^{d_y[\beta,i]}}{2^{N_{L,y}}} \right\rfloor \quad \text{and} \quad p \bmod N_{p,x} = \left\lfloor \frac{x_b \times s_x[i] \times 2^{d_x[\beta,i]}}{C_s} \right\rfloor$$

where $d_x[\beta,i]$ and $d_y[\beta,i]$ are the horizontal respectively vertical decomposition depth of component i and filter type β. $N_{L,x}$ and $N_{L,y}$ are the number of horizontal and vertical decomposition levels, and $s_x[i]$ and $s_y[i]$ are the horizontal and vertical sampling factors of component i.

The width $W_{pb}[p,b]$ of band b in precinct p is computed by $W_{pb}[p,b] = \left\lceil \frac{W_p[p]}{s_x[i] \times 2^{d_x[\beta[i]]}} \right\rceil$ for a horizontally low-pass filtered band and $W_{pb}[p,b] = \left\lfloor \left\lceil \frac{W_p[p]}{s_x[i] \times 2^{d_x[\beta[i]]-1}} \right\rceil / 2 \right\rfloor$ for a horizontally high-pass filtered band. By this definition, $W_{pb}[p,b]$ indicates the number of wavelet coefficients and also the number of quantization index magnitudes per line in precinct p and band b.

B.6 Division of precincts into lines

By the conditions in B.4, each precinct includes coefficients from at most $H_p = 2^{N_{L,y}}$ lines of wavelet coefficients. The line index λ within a precinct varies between 0 and the precinct height H_p–1:

$$\lambda \in [0, H_p - 1]$$

Band b in precinct p is included in lines $\lambda \geq L_0[p,b]$ and $\lambda < L_1[p,b]$, where L_0 is the start line and L_1 the (exclusive) end line of band b in precinct p. $L_0[p,b]$ and $L_1[p,b]$ are computed as follows:

$$L_0[p,b] = 2^{\max(N_{L,y} - d_y[i,\beta],0)} \cdot \tau_y[\beta] \quad \text{and}$$

$$L_1[p,b] = L_0[p,b] + \min\left(H_b[\beta,i] - \left\lfloor \frac{p}{N_{p,x}} \right\rfloor \times 2^{\max(N_{L,y} - d_y[i,\beta],0)}, 2^{\max(N_{L,y} - d_y[i,\beta],0)}\right)$$

where β and i are computed from b as indicated in subclause B.3. Figure B.3 provides an example how an image is divided into precincts, lines, bands and packets.

NOTE 1 By this definition, line numbers enumerate lines in the sampling grid, irrespectively of the vertical subsampling factor of a component. For vertically subsampled components, odd line numbers are excluded by means of the line inclusion flags, see subclause B.7.

NOTE 2 By these formulae, the bottommost precinct of a picture can contain bands that contain fewer lines than the bands in all other precincts. In particular, some of these bands can be even empty.

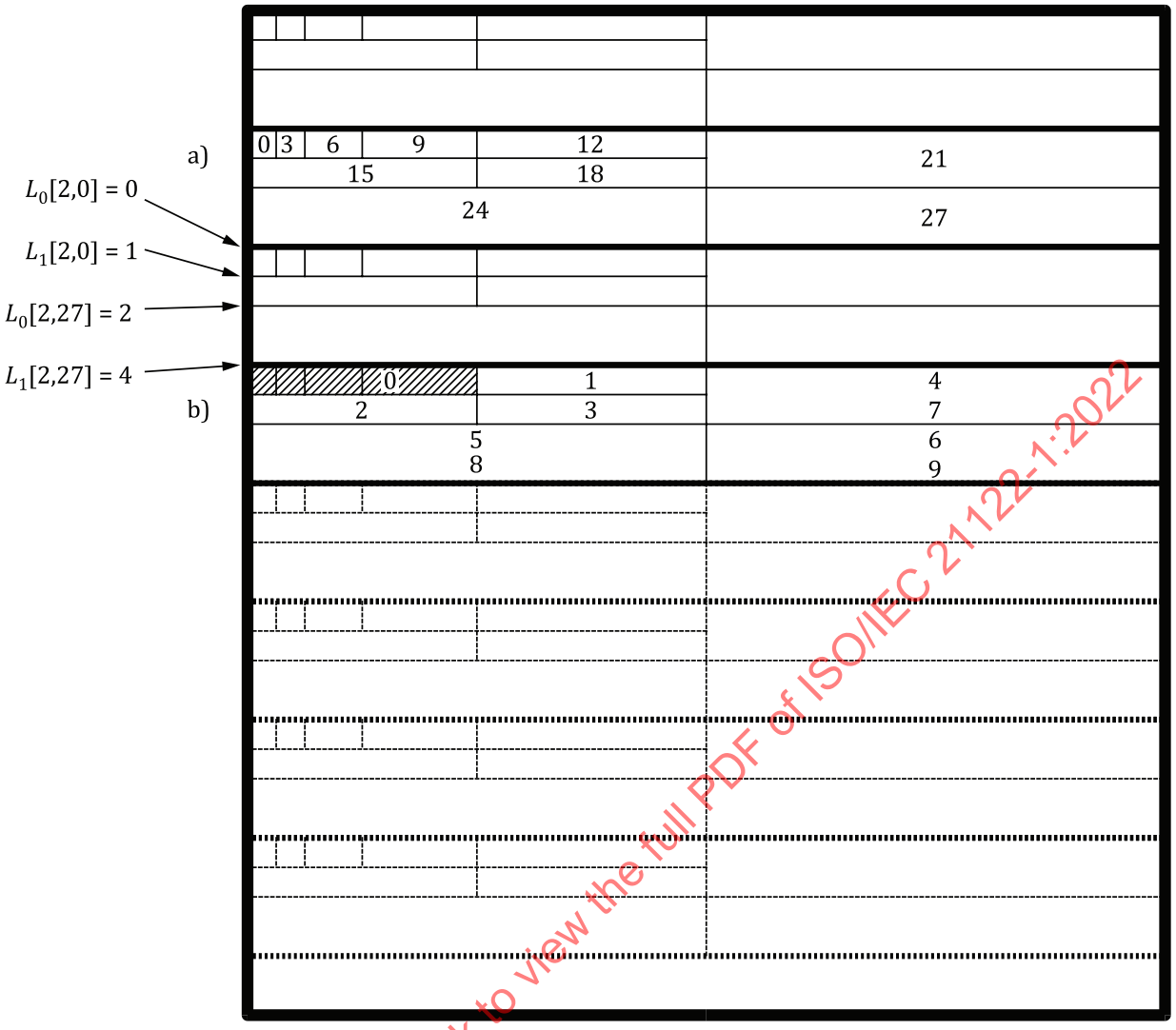


Figure B.3 — Precincts, slices and packets

NOTE 3 In Figure B.3, a 5 level horizontal and 2 level vertical decomposition using 4:4:4 or 4:2:2 sampling with a slice-height of 16 is presented. Medium lines indicate precinct boundaries, thin lines band boundaries and thick lines the image boundary. Dotted lines belong to a separate slice. The precinct denoted by a) depicts the band indices for all bands that contribute to component 0, the precinct indicated by b) depicts the packet indices. The shaded area in b) consists of a single packet. Note that some bands extend over two lines. Subpackets can extend over several bands, but include only a single line of a group of bands. Band indices for component 0 only are shown for precinct p=1, the grouping of lines of bands into packets is demonstrated for precinct p=3. Columns are disabled and precincts extend over the full image. Note further that the presence of a band in the precinct for the last precinct of the picture cannot be easily inferred from Figure B.3.

B.7 Grouping of lines and bands into packets

Each precinct p is encoded in N_{pc} packets enumerated by the packet index s, which runs from 0 to N_{pc}–1. Each packet contains entropy coded data of one or multiple bands b, but only of one single line λ of band b and precinct p. All bands within a packet are coded jointly. Whether line λ of band b in precinct p is included in packet s is encoded in the line inclusion flags $I[p,b,\lambda,s]$. This flag is non-zero in case line λ of band b and precinct p is included in packet s, and zero if it is not. As indicated by Table A.1, the line inclusion flags for precinct p are computed at the start of this precinct.

The algorithm in Table B.4 computes the line inclusion flags $I[p,b,\lambda,s]$ and the number of packets N_{pc} per precinct:

Input: Horizontal and vertical decomposition depth $N_{L,x}$ and $N_{L,y}$, number of components N_c, number of bands N_β, line start and end positions $L_0[p,b]$ and $L_1[p,b]$ for all bands, precinct index p.

Output: Line inclusion flags $I[p,b,\lambda,s]$ per precinct p, band b, line λ and packet s, number of packets $N_{pc}[p]$ for precinct p.

Table B.4 — Computation of the line inclusion flags

Name	Notes
compute_packet_inclusion(p) {	
for(b=0;b<N_L;b=b+1) {	Iterate over all band indices
for(λ=0;λ< $2^{N_{L,y}}$;λ=λ+1) {	Iterate over all lines
for(s=0;s<$N_L \times 2^{N_{L,y}}$;s=s+1) {	Iterate over all possible packet indices
I[p,b,λ,s]=0	Reset packet inclusion flag
}	End of loop over packet indices
}	End of loop over lines
}	End of loop over bands
s=0	Reset packet index
β_1=max($N_{L,x}$,$N_{L,y}$)−min($N_{L,x}$,$N_{L,y}$)+1	Number of included bands in the first packet
for(β=0;β<β_1;β= β+1) {	Loop over filter types
for(i=0;i<N_c−Sd;i=i+1) {	Loop over wavelet-decomposed components
if (b_x[β,i]) {	Check whether the band corresponding to filter type β and component i exists
b=(N_c−Sd)×β+i	Compute the band index from the filter type and the component, see subclause B.3
I[p,b,0,s]=1	β_1 bands of all components included in the first packet
}	
}	End of loop over wavelet-decomposed components
}	End of loop over wavelet filter types
for(β_0=β_1; β_0<N_β;β_0=β_0+3) {	Loop over proxy levels until all wavelet types are covered
for(λ=0;λ< $2^{N_{L,y}-d_y[0,\beta_0]}$;λ=λ+1) {	Loop over all lines within the band
for(β=β_0;β<β_0+3; β=β+1) {	Loop over all filter types within the resolution level
r=1	Indicate to create a new packet
for(i=0;i<N_c−Sd;i=i+1) {	Loop over components
if (b_x[β,i]) {	Check whether the band corresponding to filter type β and component i exists
b=(N_c−Sd)×β+i	Compute the band index from the filter type and the component, see subclause B.3
if ((λ + L_0[p,b]) umod s_y[i] == 0) {	Check whether the band is excluded due to 4:2:0 subsampling
if (λ+L_0[p,b] < L_1[p,b]) {	Check whether the line is in the precinct
s=s+r	Potentially start a new packet
I[p,b,λ+L_0[p,b],s]=1	Include the line in the precinct
r=0	Packet has been created for this band type
}	End check whether the line is included
}	End check whether subsampling allows line inclusion

Table B.4 *(continued)*

Name	Notes
}	End check whether band exists
}	End of loop over components
}	End of loop over filter types
}	End of loop over lines
}	End of loop over proxy levels
for(λ=0;λ< $2^{N_{L,y}}$;λ=λ+1) {	Loop over lines
for(i=N_c-Sd;i<N_c;i=i+1) {	Loop over components that do not participate in the wavelet decomposition
b=(N_c-Sd)×N_β+i	Compute the band from the component. The filter type is always 0
if (λ+L_0[p,b] < L_1[p,b]) {	Check whether the line is in the precinct
s=s+1	Create a new packet
I[p,b,λ+L_0[p,b],s]=1	Include the line in the precinct
}	
}	End loop over components
}	End loop over lines
N_{pc}[p]=s+1	Define the number of packets in total
}	

NOTE For *Sd*=0, the above algorithm results for 3 components, 5 horizontal and 0 vertical wavelet decomposition and 4:4:4 or 4:2:2 sampling in the line inclusion flags as listed in Table B.5, for 3 components, 5 horizontal and 1 vertical wavelet decomposition in Table B.6 and for 3 components, 5 horizontal and 2 vertical wavelet decomposition in Table B.7. For 3 components, 5 horizontal levels and 4:2:0 sampling, Table B.8 lists the lines and included bands for 1 vertical decomposition level, Table B.9 for 2 vertical decomposition levels. For *Sd*>0, the components that do not participate in the wavelet decomposition follow the regular components, with the component as fast and the line as slow variable. For *Sd*=1, 4 components and 4:4:4:4 sampling and 1 vertical decomposition level, the above algorithm results in the packet layout indicated in Table B.10, for the same configuration and 2 vertical decomposition levels in Table B.11.

Table B.5 — Line inclusion flags for zero vertical decomposition level and 4:4:4 or 4:2:2 sampling and *Sd*=0

Packet index *s*	Line number λ	Included bands
0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11) (12,13,14), (15,16,17)

Table B.6 — Line inclusion flags for one vertical decomposition level and 4:4:4 or 4:2:2 sampling and *Sd*=0

Packet index *s*	Line number λ	Included bands
0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11) (12,13,14)
1	0	(15,16,17)
2	1	(18,19,20)
3	1	(21,22,23)

Table B.7 — Line inclusion flags for two vertical decomposition levels and 4:4:4 or 4:2:2 sampling and *Sd*=0

Packet index *s*	Line number λ	Included bands
0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11)

Table B.7 *(continued)*

Packet index *s*	Line number λ	Included bands
1	0	(12,13,14)
2	1	(15,16,17)
3	1	(18,19,20)
4	0	(21,22,23)
5	2	(24,25,26)
6	2	(27,28,29)
7	1	(21,22,23)
8	3	(24,25,26)
9	3	(27,28,29)

Table B.8 — Line inclusion flags for one vertical decomposition level and 4:2:0 sampling and *Sd*=0

Packet index *s*	Line number λ of luma component	Line number λ of chroma components	Included bands
0	0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11),(12,13,14)
1	0	0	(15,16,17)
2	1	-	(18)
3	1	-	(21)

Table B.9 — Line inclusion flags for two vertical decomposition levels and 4:2:0 sampling and *Sd*=0

Packet index *s*	Line number λ of luma component	Line number λ of chroma components	Included bands
0	0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11)
1	0	0	(12,13,14)
2	1	-	(15)
3	1	-	(18)
4	0	0	(21,22,23)
5	2	2	(24,25,26)
6	2	2	(27,28,29)
7	1	-	(21)
8	3	-	(24)
9	3	-	(27)

Table B.10 — Line inclusion flags for one vertical decomposition level and 4:4:4:4 sampling with *Sd*=1

Packet index *s*	Line number λ	Included bands
0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11) (12,13,14)
1	0	(15,16,17)
2	1	(18,19,20)
3	1	(21,22,23)
4	0	(24)
5	1	(24)

Table B.11 — Line inclusion flags for two vertical decomposition levels and 4:4:4:4 sampling and *Sd*=1

Packet index *s*	Line number λ	Included bands
0	0	(0,1,2) (3,4,5) (6,7,8) (9,10,11)
1	0	(12,13,14)
2	1	(15,16,17)
3	1	(18,19,20)
4	0	(21,22,23)
5	2	(24,25,26)
6	2	(27,28,29)
7	1	(21,22,23)
8	3	(24,25,26)
9	3	(27,28,29)
10	0	(30)
11	1	(30)
12	2	(30)
13	3	(30)

B.8 Division of precinct lines into code groups

Consecutive coefficients of line λ in precinct p and band b are grouped into code groups for the purpose of joined coding. The number of coefficients within one code group is denoted by N_g and is constant throughout all bands and precincts. The first sample of the first code group in a line of a precinct corresponds to the first coefficient of that line. In case the width of the line is not a multiple of the code group size, the last code group is padded to include N_g samples. An encoder can output arbitrary values for these samples. A decoder shall ignore samples resulted from padding in all subsequent steps such as the wavelet transformation.

The number of code groups $N_{cg}[p,b]$ of precinct p and band b is computed as follows:

$$N_{cg}[p,b] = \lceil W_{pb}[p,b] \,/\, N_g \rceil$$

where $W_{pb}[p,b]$ is the width of precinct p and band b in coefficients and N_g is the size of a code group in coefficients.

B.9 Grouping of code groups into significance groups

If significance coding is enabled, multiple code groups are furthermore grouped into significance groups. A significance group comprises S_s consecutive code groups of a line λ in precinct p and band b. The first code group of the first significance group corresponds to the first code group of the precinct line. The last significance group of a precinct line may cover only a smaller number of code groups. A significance group is significant if at least one code group within the significance group contains at least one non-zero coefficient or one code group has a non-zero bitplane count prediction residual, depending on the selection of the run-mode Rm.

The number of significance groups N_s[p,b] in band b is computed as follows:

$$N_s[p,b] = \lceil W_{pb}[p,b] \,/\, (N_g \times S_s) \rceil$$

where $W_{pb}[b]$ is the width of the band b in precinct p in coefficients, N_g is the number of coefficients in a code group and S_s is the size of a significance group in code groups.

B.10 Grouping of precincts into slices

One or multiple precincts are grouped into slices. Restrictions on bitplane count decoding ensures that vertical prediction is disabled across slice boundaries; this ensures that wavelet coefficients that are part of different slices can be decoded independently of each other. Slice number t consist of

$$N_p[t] = N_{p,x} \times \begin{cases} \left(\left\lceil \frac{H_f}{H_p} \right\rceil \text{umod } H_{sl} \right), \text{if } (t+1) \times H_{sl} > \left\lceil \frac{H_f}{H_p} \right\rceil \\ H_{sl} \text{ otherwise} \end{cases}$$

precincts, such that the first slice is aligned to the top of the image, where H_{sl} is signalled in the picture header, H_f is the height of the picture, H_p is the height of a precinct in lines and $N_{p,x}$ is the number of precincts per row.

Annex C
(normative)

Entropy decoding

C.1 Entropy decoding general provisions

Encoded image data is structured in slices, see subclause B.10, where each slice includes the wavelet coefficients necessary to reconstruct a horizontal stripe of the image. Slices are represented in the codestream by slice headers and subsequent precincts, see subclause A.4.12 for the syntax of a slice header. Data following the slice header represents one or multiple precincts, where precincts are included in raster scan order, left to right, top to bottom. Precincts are not enclosed in markers. Each precinct consists of a precinct header, one or multiple packets, and optional filler bytes, see Table A.1 for details. Subclause C.2 specifies the structure of the precinct header.

Each packet s of a precinct p consists of a packet header and a packet body which itself includes multiple subpackets. Subclause C.3 specifies the structure of the packet header, and subclause C.4 the structure of the packet body. Each subpacket contributes directly or indirectly to the quantization index magnitudes $v[p,\lambda,b,x]$ and wavelet coefficient signs $s[p,\lambda,b,x]$. Some subpackets are optional; their existence is indicated by flags in the precinct header or picture header, depending on the type of the subpacket.

NOTE 1 This document does not define a mechanism to resynchronize the decoder to marker or packet boundaries. The entropy coded data can contain byte sequences that reassemble markers or marker segments. A lower level transport protocol beyond the scope of this document is needed to ensure proper resynchronization to frame or slice boundaries.

The significance subpacket includes for each significance group a single bit that, if set, indicates that all code groups within the corresponding significance groups are insignificant. A code group is insignificant if contains only zero coefficients, or its bitplane count prediction residual is zero, depending on the Run Mode flag Rm in the picture header. The significance subpacket is an optional packet that is only included if bit 1 of the bitplane count coding mode $D[p,b]$ field in the precinct header is non-zero and the raw mode override flag $D_r[p,s]$ field in the packet header is 0. The significance subpacket is defined in subclause C.5.2.

The bitplane count subpacket defines for all code groups in significant significance groups the bitplane count $M[p,\lambda,b,g]$ of a code group g. The bitplane count together with the truncation position $T[p,b]$ specifies the number of bitplanes included for all coefficients within a code group. The bitplane count subpacket is a mandatory packet that is always present. It is specified in subclause C.5.3.

The data subpacket defines the quantization index magnitudes $v[p,\lambda,b,x]$ for all code groups whose bitplane count is larger than the truncation position. If the *Fs* flag of the picture header is 0, the data subpacket also contains the signs $s[p,\lambda,b,x]$. The data subpacket is defined in subclause C.5.4.

The sign subpacket defines for all non-zero quantization index magnitudes $v[p,\lambda,b,x]$ the sign $s[p,\lambda,b,x]$ of this quantization index. The sign subpacket is an optional packet that exists only if the *Fs* flag in the picture header is non-zero. If the sign subpacket does not exist, signs are included in the data subpacket. The sign subpacket is defined in subclause C.5.5.

The bitplane count, data and sign subpackets may contain an arbitrary number of filler bytes at their end. A decoder can infer the number of filler bytes from the corresponding length field in the packet header. The value of the filler bytes shall be ignored by a decoder.

NOTE 2 The entropy coded data segment does not use markers to indicate the presence or absence of particular packet types. Instead, the picture header includes all necessary information to decide upon the presence of a particular packet.

C.2 Syntax of the precinct

A precinct is represented in the codestream by a precinct header, one or multiple packets, and – optionally – filler bytes following the entropy coded data. The amount of filler bytes following the precinct can be inferred from the $L_{prc}[p]$ field in the precinct header. Decoders shall ignore the filler bytes and skip over it, without interpreting the data stored there. Table C.1 specifies the syntax of the precinct header. Table C.2 specifies the encoding of the bitplane count coding modes.

Input: Precinct index p of the precinct whose header is to be decoded.

Output: Size of the precinct in bytes including filler bytes $L_{prc}[p]$, precinct quantization $Q[p]$, precinct refinement $R[p]$ and bitplane count coding modes $D[p,b]$ for all bands b.

Table C.1 — Precinct header syntax

Name	Notes	Size	Values
`precinct_header(p) {`			
L_{prc}[p]	Length of the entropy coded data in this precinct including filler bytes measured in bytes. The number of bytes in this field counts from the end of the precinct header of this precinct up to, but not including the first byte of the next precinct header, slice header or EOC.	u(24)	$1–(2^{20}–1)$
Q[p]	Precinct quantization. This field is input to the algorithm specified in subclause C.6.2 to select the truncation positions $T[p,b]$ of all bands of this precinct.	u(8)	0–31
R[p]	Precinct refinement. This field is input to the algorithm specified in subclause C.6.2 to select the truncation position $T[p,b]$ of all bands of this precinct.	u(8)	$0–(N_L–1)$
`for(b=0;b<N_L;b=b+1) {`	Loop over all bands		
`if (b'_x[b]) {`	Check whether band b exists		
D[p,b]	Bitplane count coding mode of band b.	u(2)	See Table C.2
`}`			
`}`	End of loop over bands		
`padding`	Pad to next byte boundary	pad(8)	0
`}`			

The $D[p,b]$ field consists of two consecutive bits per band in the precinct header. It specifies how the bitplane counts of the code groups of wavelet coefficients are encoded. Table C.2 lists valid encodings for this field in binary, where an "x" indicates a bit position whose value shall be ignored for the purpose of determining a specific function. The bitplane count encoding mode selected by the $D[p,b]$ fields can be overridden by the $D_r[p,s]$ field in the subpacket header, see Table C.3. If $D_r[p,s]$ is non-zero, bitplane counts in the corresponding subpacket are encoded in raw mode, see subclause C.6.4, regardless of the value of the $D[p,b]$ field. The same number of $D[p,b]$ fields shall be present, regardless of the values of the $D_r[p,s]$ fields and regardless whether some bands are not included at all because the last precinct is partially cut off at the bottom of the sampling grid.

The $D[p,b]$ flags shall be populated in such a way that vertical prediction is never selected for the precinct at the top of a slice; this condition ensures that wavelet coefficients in different slices can be entropy decoded independently of each other.

Table C.2 — Bitplane count coding modes

***D*[*p*,*b*]**	**Bitplane count coding mode**
x0	prediction from zero

Table C.2 *(continued)*

D[*p*,*b*]	**Bitplane count coding mode**
x1	Vertical prediction
0x	Significance coding disabled
1x	Significance coding enabled

NOTE The above table indicates that bit #1 indicates the presence of significance coding and bit #0 whether vertical or no prediction is selected.

C.3 Packet header

Data following the precinct header of precinct p consists of one or multiple packets, where each packet s represents the quantization indices of one or multiple bands b and one line λ of the precinct p and all bands within this packet, see Table A.1 for a breakdown of the syntax. A packet consists of a packet header and one or multiple subpackets. Table C.3 specifies the syntax of the packet header. Subpackets are specified in subclause C.5.

The $D_r[p,s]$ flag in the packet header indicates whether the bitplane count information of packet s in precinct p is encoded in raw. By that, $D_r[p,s]$ overrides the $D[p,b]$ mode selection in the precinct header. Regardless of the value of $D_r[p,s]$, the $D[p,b]$ flags shall be present in the precinct header. Packets encoded in the raw mode do not include significance information and the significance subpacket shall not be present for packets whose $D_r[p,s]$ field is non-zero.

In addition to the above, the following constraint shall hold if the *Rl* field of the picture header is 0: For a given precinct p and band b, the $D_r[p,s]$ flag shall be identical for all packets s that include band b within precinct p, i.e. raw and non-raw coding of bitplane counts shall not be mixed within the same band in the same precinct. Formally: for all precincts p and all packets s and s', $D_r[p,s] == D_r[p,s']$ if there is a band b and line indices λ and λ' such that $I[p,b,\lambda,s] = 1$ and $I[p,b,\lambda',s']=1$. This restriction does not apply if *Rl* is 1.

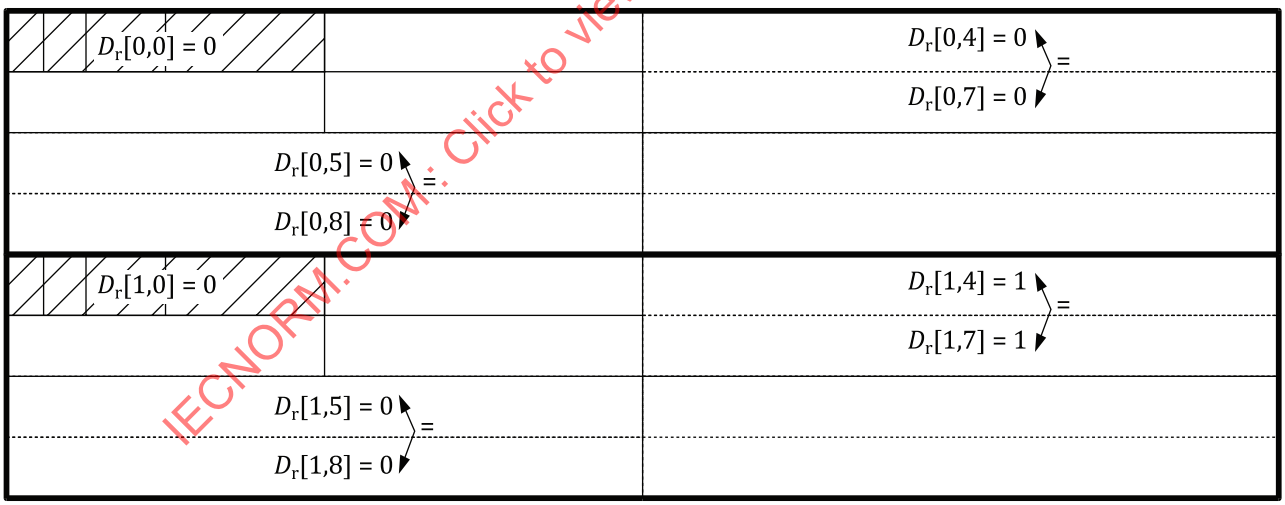


Figure C.1 — A valid selection of raw mode override flags for Rl=0

NOTE 1 Figure C.1 demonstrates a valid composition of raw-mode override flags for 5 horizontal and 2 vertical decomposition levels and two precincts for *Rl=0*. Thick lines indicate precinct boundaries, thin lines band boundaries, dotted lines packet boundaries. The shaded region to the top left is represented by a single packet. Note that raw-mode flags can vary between bands and precincts, but are identical within the same band. Not all raw mode override flags are shown.

Table C.3 specifies the syntax of the packet header.

Input: precinct index p and packet index **s** of the packet whose packet header is to be decoded.

Output: Raw mode override flag $D_r[p,s]$ for precinct p and packet s, length in bytes of the data subpacket $L_{dat}[p,s]$, length in bytes of the bitplane count subpacket $L_{cnt}[p,s]$, and length in bytes of the sign subpacket $L_{sgn}[p,s]$ of precinct p and packet s. The length of the significance packet is inferred from the number of coefficients in the bands included in packet s and is not signalled.

Table C.3 — Syntax of the packet header

Name	Notes	Size	Values
`packet_header(p,s) {`			
`ph=0`	Assume the packet header is not present		
`for(b=0;b<N_L;b=b+1) {`	Loop over all bands		
`for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {`	Loop over all lines of this band		
`if (I[p,b,λ,s]) {`	Include only if the line is present in the given band and packet, see subclause B.7.		
`ph=1`	Include the packet header		
`}`	End of line is included		
`}`	End of loop over lines		
`}`	End of loop over bands		
`if (ph==1) {`	Only include data if the packet is non-empty		
`Dr[p,s]`	Raw mode override flag. If this bit is non-zero, bitplane count information of this packet is encoded in raw mode, regardless of the value of the $D[p,b]$ flags in the precinct header.	u(1)	0,1
`if (W_f×N_c < 32752 && Lh == 0) {`	Depending on the width of the picture and the number of components, select the syntax of the header. See Table A.7 for the definition of W_f, N_c and Lh.		
$L_{dat}[p,s]$	Size of the data subpacket in bytes.	u(15)	0–32767
$L_{cnt}[p,s]$	Size of the bitplane count subpacket in bytes	u(13)	0–8191
$L_{sgn}[p,s]$	Size of the sign subpacket in bytes if Fs=1. If Fs=0, this field shall be present, but is ignored. Fs is specified in the picture header, see Table A.7.	u(11)	0–2047
`} else {`	End of short packet header		
$L_{dat}[p,s]$	Size of the data subpacket in bytes.	u(20)	0–1048575
$L_{cnt}[p,s]$	Size of the bitplane count subpacket in bytes	u(20)	0–1048575
$L_{sgn}[p,s]$	Size of the sign subpacket in bytes if Fs=1. If Fs=0, this field shall be present, but is ignored. Fs is specified in the picture header, see Table A.7.	u(15)	0–32767
`}`	End of condition for packet header size		
`}`	End of test for non-empty packet		
`}`			

NOTE 2 In case a component is not vertically subsampled or excluded from the wavelet transformation by means of the CWD marker, the data subpackets of such a component can grow larger than 32768 bytes or the sign subpacket can grow larger than 2048 bytes, even if $W_f \times N_c < 32752$. It is thus advisable to enforce long headers with Lh=1 if the codestream contains components that are not vertically decomposed or do not participate in the wavelet filtering process, and if $W_f \times N_c \geq 16376$.

C.4 Packet body

Table C.4 specifies the syntax of the packet body precinct p and packet s. The packet body of packet s in precinct p consists of multiple subpackets, each of which contributes directly or indirectly to the bitplane counts $M[p,\lambda,b,g]$, the quantization index magnitudes $v[p,\lambda,b,x]$ or quantization index signs $s[p,\lambda,b,x]$ of a single line λ and one or multiple bands b of precinct p.

Input: Precinct index p and packet index s.

Output: Bitplane counts $M[p,\lambda,b,g]$, quantization index magnitudes $v[p,\lambda,b,x]$ and quantization index signs $s[p,\lambda,b,x]$ for precinct p, line λ, and all bands b in subpacket s.

Table C.4 — Syntax of the packet body

Name	Notes	Reference
`packet_body(p,s) {`		
`unpack_significance(p,s);`	Decode significance information. The size of the significance sub-packet is not included in the packet header and can be inferred from the size of the line and the band.	Table C.5
`unpack_bitplane_count(p,s);`	Decode bitplane count information. This subpacket includes $L_{cnt}[p,s]$ bytes.	
`unpack_data(p,s);`	Decode wavelet magnitude data. This subpacket includes $L_{dat}[p,s]$ bytes.	Table C.8
`if (Fs==1) {`	The sign subpacket is only included if sign coding is enabled in the picture header, see Table A.7 for the definition of *Fs*.	
`unpack_signs(p,s);`	Decode wavelet magnitude data. This subpacket includes $L_{sgn}[p,s]$ bytes.	Table C.9
`}`	End of if sign packing enabled	
`}`		

C.5 Subpackets

C.5.1 Nomenclature

The entropy coded data segment following the packet header is transmitted in multiple *subpackets*. Each subpacket contains data of a specific type that is relevant to one line but one or multiple bands of the precinct indicated by the packet header. Depending on configuration, not all subpackets may be present.

C.5.2 Significance subpacket

Table C.5 specifies the syntax of the significance subpacket. This subpacket includes for every significance group of code groups one bit that identifies whether all code groups in the significance group are insignificant. The bitplane count subpacket does not include information for insignificant significance groups and the bitplane counts of the code groups within such significance groups are inferred. This subpacket is optional. It is only included if bit #1 of the $D[p,b]$ field of the precinct header

is set to 1 and the raw mode override flag $D_r[p,s]$ is set to 0. See subclause C.3 for the specification of the precinct header.

NOTE The packet header does not include the size of the significance subpacket, it can be inferred from the included bands.

Input: Precinct *p* and packet *s* whose significance data is to be decoded

Output: Significance flags $Z[p,\lambda,b,j]$ of all significance groups and all bands of the given precinct *p* and packet *s*.

Table C.5 — Syntax of the significance subpacket

Name	Semantics	Size	Values
`unpack_significance(p,s) {`			
`for(b=0;b<N_L;b=b+1) {`	Loop over all bands		
`for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {`	Loop over all lines of this band		
`if (I[p,b,λ,s]) {`	Include only if the line is present in the given band and packet, see subclause B.7.		
`if (D_r[p,s] == 0) {`	Include only if the raw override flag is not set		
`if (D[p,b] & 2) {`	Significance information is only present if indicated by bit #1 of $D[p,b]$ in the precinct header is set.		
`for(j=0;j<N_s[p,b];j = j + 1) {`	Loop over all significance groups of this precinct, band and line. A definition of $N_s[p,b]$ is given in subclause B.9.		
`Z[p,λ,b,j]`	Significance information of this significance group	u(1)	0,1
`}`	End of loop over significance groups		
`}`	End of significance coding enabled		
`}`	End of raw override is not set		
`}`	End of line included		
`}`	End of loop over lines		
`}`	End of loop over bands		
`Padding`	Pad to the next byte boundary	pad(8)	
`}`			

C.5.3 Bitplane count subpacket

C.5.3.1 Purpose of the Bitplane count subpacket

The Bitplane count subpacket decodes to the bitplane counts of the code groups of a packet *s* of precinct *p*. The syntax of the packet depends on the bitplane count coding mode $D[p,b]$ of the precinct header and the raw mode override flag $D_r[p,s]$ signalled in the packet header. Additional constraints apply to the selection of the bitplane count coding mode $D[p,b]$ and the raw mode override flag $D_r[p,s]$. Which contraints apply depend on the raw-mode selection per packet flag *Rl* of the picture header, see subclause A.4.4. In case *Rl*=0, the constraints indicated in subclause C.5.3.2 apply. In case *Rl*=1, the constraints indicated in subclause C.5.3.3 apply. Subclause C.5.3.4 specifies an algorithm that tests the correctness of the mode selection, and codestreams shall be constructed in such a way that this

algorithm succeeds. Subclause C.5.3.5 specifies the syntax and the decoding algorithm for the bitplane count subpacket.

NOTE The purpose of these constraints are to ensure an upper bound for the buffer size a decoder has to reserve for entropy-coded bitplane count data. An encoder can always satisfy the constraints by selecting the raw mode for the bitplane count coding mode if the size of the bitplane count subpacket becomes too large.

C.5.3.2 Bitplane count mode selection for Rl=0

In case *Rl*=0, the codestream shall be constructed in such a way that for all bands *b*, the sum of the sizes of all bitplane count subpackets and significance subpackets contributing to *b* is at most as large as the sum of all sizes of encoding the bitplane count of the same subpackets in the raw mode.

Formally: Let $L_{\mathrm{cnt}}[p,s]$ be the size of the bitplane count subpacket of precinct p and packet s in bytes defined in Table C.3. Let $L_{\mathrm{sig}}[p,s]$ be the size of the significance subpacket of precinct p and packet s, or 0 if no significance data is included.

While $L_{\mathrm{sig}}[p,s]$ is not explicitly signalled, it can be inferred from the size of the bands contributing to s, either by the number of bytes generated by the algorithm specified in Table C.5, or equivalently by

$$L_{\mathrm{sig}}[p,s]=\left\lceil \frac{\sum_{b,\lambda} I[p,b,\lambda,s]\times(1-D_{\mathrm{r}}[p,s])\times(D[p,s]\gg 1)\times N_{\mathrm{s}}[p,b]}{8} \right\rceil$$

Let $L'_{\mathrm{tot}}[p,b]$ be the sum of sizes of all bitplane count and significance subpackets contributing to band b in precinct p:

$$L'_{\mathrm{tot}}[p,b] := \sum_{s=0}^{N_{\mathrm{pc}}-1} \sum_{\lambda=L_0[p,b]}^{L_1[p,b]-1} I[p,b,\lambda,s]\times\left(L_{\mathrm{cnt}}[p,s]+L_{\mathrm{sig}}[p,s]\right)$$

Let $L_{\mathrm{tot}}^{\mathrm{raw}}[p,b]$ be the amount of bytes required to encode the bitplane count of band b in precinct p in the raw coding mode:

$$L_{\mathrm{tot}}^{\mathrm{raw}}[p,b] := \sum_{s=0}^{N_{\mathrm{pc}}-1} \sum_{\lambda=L_0[p,b]}^{L_1[p,b]-1} I[p,b,\lambda,s]\times \sum_{b'=0}^{N_{\mathrm{l}}-1} \left\lceil \frac{I[p,b',\lambda,s]\times N_{\mathrm{cg}}[p,b']\times B_r}{8} \right\rceil$$

Then, the codestream shall be constructed in such a way that for all b and p, $L'_{\mathrm{tot}}[p,b] \le L_{\mathrm{tot}}^{\mathrm{raw}}[p,b]$ holds.

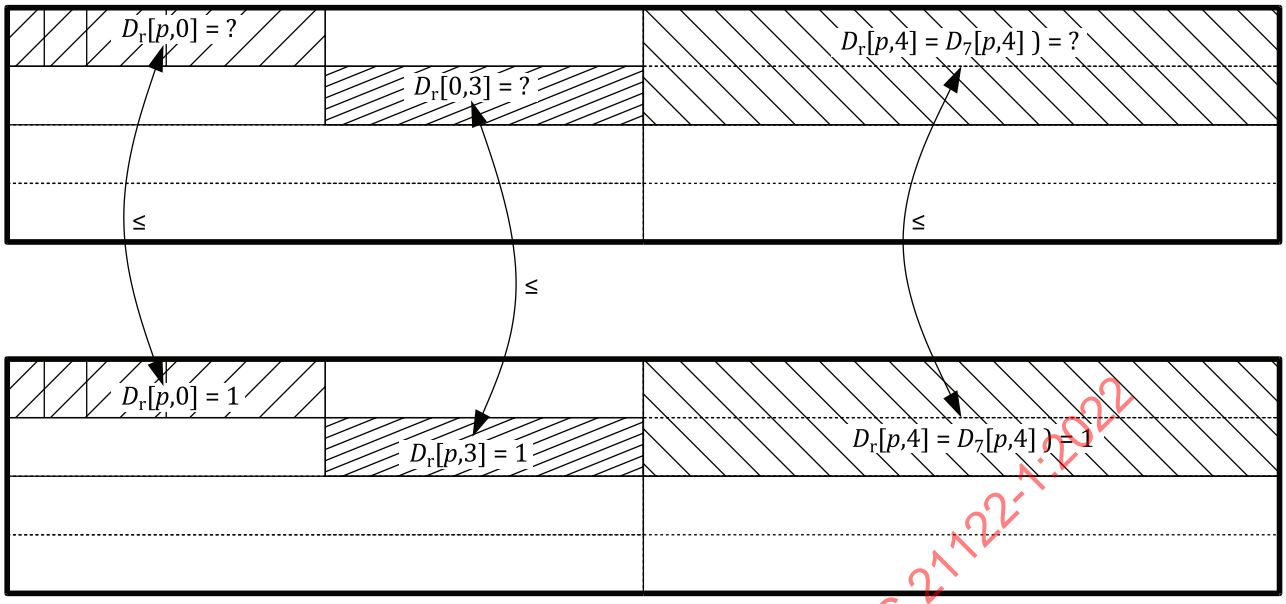


Figure C.2 — Rate constraints for band and packet mode selections

NOTE Figure C.2 demonstrates which comparisons are made to test the validity of a mode decision for a single precinct p with a 5 level horizontal, 2 level vertical wavelet decomposition. Thick lines represent precinct boundaries, thin lines band boundaries and dotted lines packet boundaries. The shaded area to the top left is encoded in a single packet. The overall rate of the bitplane count subpackets in the shaded areas is computed once for all packets covering the area encoded with the bitplane count coding mode $D[p,b]$ and $D_r[p,s]$ as given (top row) and once in raw mode $D_r[p,s]$=1 (bottom row). Areas over which the rate is summed are defined in such a way that the raw mode flags are consistent with the condition specified in subclause C.3. The bitplane count coding modes $D[p,b]$ and raw mode override flags $D_r[p,s]$ in a codestream are populated in such a way that the rate in any of the shaded areas is not larger than the rate of the same area in the bottom row. A trivial, but suboptimal way to satisfy this constraint would be to select $D_r[p,s]$=1 for all packets.

C.5.3.3 Bitplane count mode selection for *Rl*=1

In case *Rl*=1, the bitstream shall be constructed in such a way that that for all bands b and all lines λ, the sum of the sizes of all bitplane count subpackets and significance subpackets contributing to b in line λ is at most as large as the sum of all sizes of encoding the bitplane count of the same subpackets in the raw mode.

Formally: Let $L_{cnt}[p,s]$ and $L_{sig}[p,s]$ be defined as in subclause C.5.3.2.

Let $L'^{l}_{tot}[p,s]$ be the sum of sizes of all bitplane count and significance subpackets contributing to precinct p and packet s:

$$L'^{l}_{tot}[p,s] := L_{cnt}[p,s] + L_{sig}[p,s]$$

Let $L^{l,raw}_{tot}[p,s]$ be the amount of bytes required to encode the bitplane count of precinct p and packet s in the raw coding mode:

$$L^{l,raw}_{tot}[p,s] := \sum_{b=0}^{N_l-1} \sum_{\lambda=L_0[p,b]}^{L_1[p,b]-1} \left\lceil \frac{I[p,b,\lambda,s] \times N_{cg}[p,b] \times B_r}{8} \right\rceil$$

Then, the codestream shall be constructed in such a way that for all p and s, $L'^{l}_{tot}[p,s] \le L^{l,raw}_{tot}[p,s]$ holds.

C.5.3.4 Validation algorithm for bitplane count and raw mode override selection

Table C.6 specifies an algorithm that checks the validity of the encoding of a precinct p by checking the above condition for all bands *b* in *p*:

Input: Precinct index *p*

Output: An indicator *valid* that is 1 in case the precinct mode selection is valid, or 0 in case it is invalid.

Table C.6 — Testing the validity of a precinct encoding

Name	Notes
is_encoding_valid(p){	
valid=1	Assume validity
if (Rl==0) {	Check for packet-based raw-mode switch
for(b=0;b<N_L;b=b+1) {	Loop over all subbands of the precinct
rawsize=0	Number of bits required to encode this band in raw mode
bytesize=0	Number of bytes
for(s=0;s<N_{pc};s=s+1) {	Loop over all packets in the precinct
for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {	Loop over all lines in the band
if (I[p,b,λ,s]) {	Include only if the band *b* is present in the given packet *s*, line and precinct, see subclause B.7.
bytesize = bytesize + L_{cnt}[p,s]	Include number of bytes required for bitplane count coding
bytesize = bytesize + L_{sig}[p,s]	Include number of bytes required for significance coding
for(b'=0;b'<N_L;b'= b'+1) {	Loop over all bands that contribute to the same packet *s* band b is part of
if (I[p,b',λ,s]) {	Only if band *b'* is included in the same subpacket
rawsize = rawsize + B_r×N_{cg}[p,b']	Reserve B_r bits per included bitplane count for each line in this precinct. $N_{cg}[p, b']$ is specified in subclause B.8.
}	End of band *b'* is included in subpacket *s*
}	End of loop over bands *b'*
}	End of if line included
}	End of loop over lines
}	End of loop over subpackets
if (bytesize > ⌈rawsize/8⌉) {	Check buffer limit condition
valid=0	Invalid if buffer size constraint violated
}	End of check whether bitrate constraint is satisfied
}	End of loop over bands
} else {	End of band-based raw-mode switch, start of packet-based switch.
for(s=0;s<N_{pc};s=s+1) {	Loop over all packets in the precinct
rawsize = 0	
bytesize = L_{cnt}[p,s] + L_{sig}[p,s]	Include number of bytes required for bitplane count coding and significance coding
for(b=0;b<N_L;b=b+1) {	Loop over all subbands of the precinct
for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {	Loop over all lines in the band
if (I[p,b,λ,s]) {	Only if band b is included in the subpacket

Table C.6 *(continued)*

Name	Notes
rawsize = rawsize + B_r×N_{cg}[p, b]	Reserve B_r bits per included bitplane count for each line in this precinct. $N_{cg}[p, b]$ is specified in subclause B.8.
}	End of check for band inclusion
}	End of loop over lines
}	End of loop over bands
if (bytesize > ⌈rawsize/8⌉) {	Check buffer limit condition
valid=0	Invalid if buffer size constraint violated
}	
}	End of loop over all packets
}	End of line-based raw-mode switch
}	

C.5.3.5 Bitplane count subpacket syntax

Table C.7 specifies the syntax of the bitplane count subpacket depending on $D[p,b]$ and $D_r[p,s]$ and gives reference to the corresponding subclauses.

NOTE The number of filler bytes at the end of the bitplane count subpacket can be inferred from the $L_{cnt}[p,s]$ field of the packet header.

Input: precinct and packet whose bitplane counts are to be decoded, significance flags $Z[p,\lambda,b,x]$ of the line and precinct if significance coding is enabled.

Output: Bitplane counts $M[p,\lambda,b,g]$ in all bands of the precinct p and packet s.

Table C.7 — Syntax of the Bitplane count subpacket

Name	Notes	Size	Reference
unpack_bitplane_count(p,s){			
for(b=0;b<N_L;b=b+1) {	Loop over all bands		
for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {	Loop over all lines of this band		
if (I[p,b,λ,s]) {	Include only if the line is present in the given band and packet, see subclause B.7.		
if (Dr[p,s]==1){	Detect whether raw coding is enabled for this packet		
unpack_raw(p,b,λ);	Decode with the raw coding mode	Variable	Subclause C.6.4
} else if ((D[p,b] & 1) == 0) {	Select the bitplane count coding mode		
unpack_nopred(p,b,λ);	No prediction with or without sig-flags	Variable	Subclause C.6.6
} else {			
unpack_vertical(p,b,λ);	Vertical prediction with or without sig-flags	Variable	Subclause C.6.5
}	End of bitplane count coding mode selection		
}	End of if line is present in subpacket and band		
}	End of loop over lines		
}	End of loop over bands		
Padding	Pad to the next byte boundary	pad(8)	
filler bytes	Arbitrary number of filler bytes	fill()	
}			

C.5.4 Data subpacket

Table C.8 specifies the syntax of the data subpacket. This subpacket includes the coefficient data of all significant code groups of a given precinct and line within a precinct. It also requires the bitplane count of each code group and the truncation position of each subband.

NOTE The number of filler bytes at the end of the data subpacket can be inferred from the $L_{dat}[p,s]$ field of the packet header.

Input: precinct and packet whose coefficient data is to be decoded, bitplane counts of all code groups of all bands of the given line and precinct, truncation positions of all lines and bands of the given precinct and line. The truncation positions $T[b,p]$ are computed from the information in the precinct header specified in subclause C.2 and the weights table specified in subclause A.4.6 according to the algorithm given in subclause C.6.2.

Output: Magnitudes of the quantization index magnitudes $v[p,l,b,x]$ in all bands of the precinct p and packet s, and if sign packing is disabled, additionally quantization index signs $s[p,l,b,x]$ of all bands in the given precinct p and packet s.

Table C.8 — Syntax of the data subpacket

Name	Notes	Size	Values
unpack_data(p,s) {			
for(b=0;b < N_L;b=b+1) {	Loop over all bands of the precinct		
for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {	Loop over all lines of this band		
if (I[p,b,λ,s]) {	Include only if the line is present in the given band and packet, see subclause B.7.		
for(g=0;g<N_{cg}[p,b];g=g+1) {	Include data for all groups in this precinct and line.		
v[p,λ,b,N_g×g+k] = 0	Reset quantization index magnitude		
if (M[p,λ,b,g]>T[p,b]) {	Include only signs if a non-zero number of bit-planes is included.		
if (Fs == 0) {	Check whether the sign subpacket is disabled		
for(k=0;k<N_g;k=k+1) {	Loop over all members of the code group. The definition of the group size N_g is specified in subclause B.8.		
s[p,λ,b,N_g×g+k]	Sign bit of the coefficient in the current band, line and group	u(1)	0,1
}	End of loop over coefficients		
}	End of sign inclusion		
for(i=M[p,λ,b,g]-T[p,b]-1; i≥0;i=i-1) {	Loop over all bit positions		
for(k=0;k<N_g;k=k+1) {	Loop over all members of the code group		
D	Binary data of the quantization index magnitude in the precinct, line, band and group	u(1)	0,1

Table C.8 *(continued)*

Name	Notes	Size	Values
v[p,λ,b,N_g×g+k]=v[p,λ,b,N_g×g+k]+(d<<i)	Set the corresponding bit in the quantization index magnitude		
}	End of loop over code group members		
}	End of loop over bitplanes		
}	End of non-zero number of bitplanes included		
}	End of loop over code groups		
}	End of line and band included in subpacket		
}	End of loop over lines		
}	End of loop over bands		
Padding	Pad to the next byte boundary	pad(8)	
filler bytes	Arbitrary number of filler bytes	fill()	
}			

C.5.5 Sign subpacket

Table C.9 specifies the syntax of the sign subpacket. This subpacket includes the sign information of all coefficients of all code groups of a given precinct and line within a precinct. This subpacket shall only be present if the sign packing flag *Fs* specified in subclause A.4.3 is set to 1.

NOTE 1 The number of filler bytes at the end of the sign subpacket can be inferred from the $L_{sgn}[p,s]$ field of the packet header.

Input: precinct and packet whose sign data is to be decoded, decoded coefficient magnitudes of the precinct and subpacket.

Output: array of signs $s[p,\lambda,b,x]$ of all bands in the given precinct and packet, coefficient array of all coefficients in the precinct and line.

Table C.9 — Syntax of the sign subpacket

Name	Semantics	Size	Values
unpack_signs(p,s) {			
for(b=0;b<N_L;b=b+1) {	Loop over all bands		
for(λ=L_0[p,b];λ<L_1[p,b];λ=λ+1) {	Loop over all lines of this band		
if (I[p,b,λ,s]) {	Include only if the band is present in the given line and packet, see subclause B.7.		
for(g=0;g<N_{cg}[p,b];g=g+1) {	Include data for all groups in this precinct and line. See B.8 for a definition of N_{cg}.		
for(k=0;k<N_g;k=k+1) {	Iterate over all members of the code group.		
if (v[p,λ,b,N_g×g+k]!=0) {	Only include sign information if the quantization index magnitude is non-zero		
s[p,λ,b,N_g×g+k]	Sign bit of non-zero quantization index magnitude	u(1)	0,1
}	End of non-zero code group		

Table C.9 *(continued)*

Name	Semantics	Size	Values
}	End of loop over coefficients		
}	End of loop over groups		
}	End of line and band included in subpacket		
}	End of loop over lines		
}	End of loop over bands		
Padding	Pad to the next byte boundary	pad(8)	
filler bytes	Arbitrary number of filler bytes	fill()	
}			

NOTE 2 As the data subpacket always transmits coefficients in groups of 4, it can happen that it includes meaningless coefficients near the right edge of a wavelet band. By the above table, the sign subpacket includes sign bits even for such meaningless coefficients whenever they are non-zero. It is advisable, though not necessary, to force such meaningless coefficients to 0 at the encoder side.

C.6 Bitplane count decoding

C.6.1 Bitplane count decoding general provisions

The bitplane count decoding process decodes the bitplane counts $M[p,\lambda,b,g]$ from the contents of the Bitplane count subpacket by a process specified in subclause C.5.3 that depends on the bitplane count coding mode $D[p,b]$ in the precinct header and the raw mode override flag $D_r[p,s]$ in the packet header. The decoding process may optionally predict bitplane counts vertically, and may optionally employ significance flags to skip over insignificant code groups.

The bitplane count decoding process requires furthermore the truncation position $T[p,b]$, which defines the bitplane at which transmission of coefficient data stops, and hence indirectly determines the quantization step size. The computation of the truncation position is specified in subclause C.6.2.

Vertical prediction modes also require access to the bitplane counts $M_{top}[p,\lambda,b,g]$ and truncation position $T_{top}[p,b]$ of the line directly above the current line within the same band. The computation of $M_{top}[p,\lambda,b,g]$ and $T_{top}[p,b]$ is specified in subclause C.6.3. The codestream shall be constructed in such a way that vertical prediction is never selected as bitplane count coding mode for the topmost lines of the topmost precinct of a slice or the image.

NOTE The above requirement ensures that wavelet coefficients within different slices can be decoded independently of each other.

C.6.2 Computation of the truncation position

Table C.10 specifies the computation of the truncation position $T[b,p]$ of band b and precinct p from the precinct quantization $Q[p]$ and precinct refinement $R[p]$, both specified in the precinct header specified in subclause C.2, and the band priority $P[b]$ defined by the weights table specified in subclause A.4.11. The truncation position $T[b,p]$ is required for multiple inverse prediction processes as well as for the inverse quantization defined in subclause D.1

Input: Band index b, precinct index p, precinct quantization $Q[p]$, precinct refinement $R[p]$ and band priority $P[b]$.

Output: Truncation position $T[b,p]$ of band b in precinct p.

Table C.10 — Computation of the truncation position

Syntax	Notes
`compute_truncation(p,b) {`	
`if (P[b]<R[p]) {`	Compare the priority of the band $P[b]$ as specified in the weights table with the refinement threshold $R[p]$ in the precinct header
`r = 1`	An additional bitplane is included for bands with priorities below the refinement threshold
`} else {`	
`r = 0`	No refinement otherwise
`}`	
`T[b,p]=clamp(Q[p]-G[b]-r,0,15)`	Compute the truncation position as the precinct quantization minus the band gain from the Weights Table, minus the number of additional refinement bitplanes, then clamp to the valid range.
`}`	

C.6.3 Computation of the vertical bitplane count predictor and truncation position predictor

Vertical prediction modes require an entire row of bitplane counts above the current line as the source of the prediction. In addition, the truncation position of the line above is also required. The codestream shall be constructed in such a way that vertical prediction is not selected at the first line of a slice, and hence in particular not at the top of the image.

The process specified in Table C.11 computes from a given precinct p, line λ and band b the corresponding vertical predictor $M_{\text{top}}[p,\lambda,b,g]$ and the vertical truncation predictor $T_{\text{top}}[p,\lambda,b,g]$. For that, it requires the first line $L_0[p,b]$ and the last line $L_1[p,b]$ of band b and precinct p, where $L_0[p,b]$ and $L_1[p,b]$ are specified in subclause B.6.

Input: Precinct index p, line index λ, band index b, bitplane counts $M[p,\lambda,b,g]$ of precinct p and precinct $p-N_{p,x}$, first line $L_0[p,b]$ and last line $L_1[p,b]$ of band b of precinct p. Let s_y be the vertical subsampling factor $s_y[i]$ of the component the band b is part of.

Output: Vertical predictors $M_{\text{top}}[p,\lambda,b,g]$ of all code groups g of precinct p, line λ and band b; truncation position predictor $T_{\text{top}}[b,p]$

NOTE Vertical prediction cannot be selected at the start of the slice, and hence at the top of the image. Hence, the algorithm as specified here always accesses bitplane counts within the current slice and within the image.

Table C.11 — Computation of the vertical bitplane count predictor

Syntax	Notes
`compute_predictor(p,b,λ) {`	
`for(g=0;g<N_cg[b];g=g+1) {`	Loop over all code groups of this band
`if (λ-s_y < L_0[p,b]) {`	Check whether the given line is the first line of the band in the precinct
`M_top[p,λ,b,g]=M[p-N_p,x,L_1[p,b]-s_y,b,g]`	Predict from the last line of the precinct above if λ is the top line of the band in the precinct
`} else {`	
`M_top[p,λ,b,g]=M[p,λ-s_y,b,g]`	Precinct from the line above λ if this line still in the same precinct
`}`	
`}`	End of loop over all code groups
`if (λ-s_y < L_0[p,b]) {`	Check whether the given line is the first line of the band in the precinct

Table C.11 *(continued)*

Syntax	Notes
T_{top}[p,b]=T[p-$N_{p,x}$,b]	Predict from the precinct above if λ is the top line of the band in the current precinct
} else {	
T_{top}[p,b]=T[p,b]	Precinct from the line above if still in the precinct
}	
}	

C.6.4 Bitplane count decoding for the raw mode.

The syntax of the bitplane count subpacket specified in this subclause is selected if $D_r[p,s]$ in the packet header is 1, indicating the raw mode. The bitplane count subpacket contains in this case bitplane counts $M[p,\lambda,b,g]$ directly, using B_r bits per code group. Table C.12 specifies the decoding of the bitplane counts in the raw mode.

Input: precinct index p, band b, and line index λ whose bitplane counts are to be decoded.

Output: Bitplane counts $M[p,\lambda,b,g]$ for all other code groups g of precinct p, band b and line λ.

Table C.12 — Raw mode

Name	Notes	Size	Values
unpack_raw(p,b,λ) {			
for(g=0;g<N_{cg}[p,b];g=g+1) {	Include predicted bitplane counts for all code groups. N_{cg} is specified in subclause B.8.		
M[p,λ,b,g]	Bitplane count encoded in B_r bits	u(B_r)	0–(2^{Br}-1)
}	End of loop over code groups		
}			

C.6.5 Differential bitplane count decoding for vertical prediction

The syntax of the bitplane count subpacket specified in this subclause is selected if the bitplane count coding mode $D[p,b]$ in the precinct header indicates vertical prediction and the raw mode override flag $D_r[p,s]$ in the packet header is 0. Table C.13 specifies how to decode bitplane counts in the vertical mode. The bitplane count subpacket contains in this case bitplane count prediction residuals which are used to recover the bitplane counts from the residuals by inverse vertical prediction. The codestream shall always be constructed in such a way that the bitplane counts $M[p,\lambda,b,g]$ are between 0 and (2^{Br}-1).

Input: precinct index p, band b and line index λ whose magnitude data is to be decoded, significance information of the precinct and line whose magnitude information is to be decoded. Significance flags $Z[p,\lambda,b,g]$ in case significance coding is enabled by the bitplane count coding mode $D[p,b]$.

Output: Bitplane counts $M[p,\lambda,b,g]$ for all code groups g of precinct p, band b and line λ.

Table C.13 — Vertical mode

Name	Notes	Size	Values
unpack_vertical(p,b,λ) {			
compute_predictor(p,b,λ)	Compute the prediction values, see subclause C.6.3		
for(g=0;g<N_{cg}[p,b];g=g+1) {	Include predicted bitplane counts for all code groups. N_{cg} is specified in subclause B.8.		
t = max(T[p,b],T_{top}[p,b])	Compute effective truncation position for prediction		

Table C.13 *(continued)*

Name	Notes	Size	Values
m_{top} = max(M_{top}[p,λ,b,g],t)	Compute predictor		
if ((D[p,b] & 2) == 0 || Z[p,λ,b, ⌊g/S_s⌋] == 0) {	Decode prediction residual only if either significance information was not included, or the corresponding significance group was signalled as significant		
Δm=vlc(m_{top},T[p,b])	Decode prediction residual encoded with variable length code	vlc(m_{top},T[p,b])	
} else {			
if (Rm == 0) {	Test the run mode		
Δm=0	Non-significant groups have a zero bitplane count prediction residual.		
} else {			
Δm=T[p,b]−m_{top}	Non-significant groups have a bitplane count of T[p,b].		
}	End of run mode selection		
}	End of test on significance of code group		
M[p,λ,b,g] = m_{top} +Δm	Predict from m_{top}		0–(2^{Br}-1)
}	End of loop over code groups		
}			

C.6.6 Variable length bitplane count decoding without prediction

The syntax of the bitplane count subpacket specified in this subclause is selected if D[p,b] in the precinct header indicates no prediction and the raw mode override flag D_r[p,s] in the packet header is 0. Table C.14 specifies how to decode bitplane counts in the no-prediction mode. The bitplane count subpacket contains in this case variable length encoded bitplane counts. The codestream shall always be constructed in such a way that the bitplane counts M[p,λ,b,g] are between 0 and (2^{Br}–1).

Input: precinct index p, band b and line index λ whose magnitude data is to be decoded, significance information of the precinct and line whose magnitude information is to be decoded. Significance flags Z[p,λ,b,g] in case significance coding is enabled by the bitplane count coding mode D[p,b].

Output: Bitplane counts M[p,λ,b,g] for all code groups g of precinct p, band b and line λ.

Table C.14 — No-prediction mode

Name	Notes	Size	Values
unpack_nopred(p,b,λ) {			
for(g=0;g<N_{cg}[p,b];g=g+1) {	Include predicted bitplane counts for all code groups. N_{cg} is specified in subclause B.8.		
m_{top} = T[p,b]	Set predictor to the truncation position		
if ((D[p,b] & 2) == 0 || Z[p,λ,b, ⌊g/S_s⌋] == 0) {	Decode prediction residual only if either significance information was not included, or the corresponding significance group was signalled as significant		
Δm=vlc(m_{top},T[p,b])	Decode prediction residual encoded with variable length code	vlc(m_{top},T[p,b])	
} else {			
Δm=0	Non-significant groups have bitplane count T[p,b].		
}	End of significance included		

Table C.14 *(continued)*

Name	Notes	Size	Values
m = m_{top} +Δm	Predict from m_{top}		
M[p,λ,b,g] = m			0–(2^{Br}-1)
}	End of loop over code groups		
}	End of test on significance of code group		

C.7 Elementary variable length coding and decoding primitives

C.7.1 Variable length decoding primitive

Table C.15 specifies the variable length decoder vlc() primitive which decodes a signed quantity in the context (r,t) of a predictor *r* and a truncation position *t*. This coding primitive is used throughout this annex. A codestream shall not contain more than 32 1-bits as input for the vlc decoder. Detecting such a condition indicates that the decoder has lost synchronization with the source. This establishes an error condition whose handling is beyond the scope of this document.

Input: A predictor *r* and a truncation position *t*.

Output: a signed quantity *x*.

Table C.15 — Decoding a signed quantity with vlc

Syntax	Semantics	Size	Values
vlc(r,t) {			
θ=max(r-t,0)	Compute the threshold for the alphabet switch		
x=0	Reset the bitcounter:		
do {			
b		u(1)	0,1
if (b) {			
x = x+1	Count the number of 1-bits		
}			
} while(b && x < 32)	Repeat as long as 1-bits are found in the stream		
if (x ≥ 32) {			
error()	A codestream shall not contain more than 32 consecutive 1-bits		
}			
if (x > 2×θ) {	Check whether this is the unary sub-alphabet		
return x-θ	If so, decode unary subalphabet		
} else if (x > 0) {	Check for non-zero symbol, signed sub-alphabet		
if (x & 1) {	Check for an odd codeword		
return $-\lceil x/2 \rceil$	Return a negative value for odd codewords		
} else {			
return $\lfloor x/2 \rfloor$	Return a positive value for an even codeword		
}			
} else {			
return 0	Return zero for a zero codeword		
}			
}			

C.7.2 Variable length encoding primitive

Table C.16 provides guidance on the implementation of an algorithm that inverts the vlc() decoder primitive and encodes a signed quantity *x* in the context of a predictor *r* and a truncation position *t*.

Input: A predictor *r*, a truncation position *t* and a signed quantity *x* to be encoded.

Output: a sequence of bits encoding *x* given the context consisting of *r* and *t*

Table C.16 — Encoding a signed quantity with vlc

Syntax	Semantics
`vlc_encode(x,r,t) {`	Encode x in the context of *r* and *t*
`θ = max(r-t,0)`	Compute the threshold for the alphabet switch
`if (x > θ) {`	Check for the unary sub-alphabet case
`n = x + θ`	Compute the number of one-bits to write in the unary case
`} else {`	Instead in the binary sub-alphabet case
`x = x × 2`	Reserve two bits per symbol in the binary sub-alphabet
`if (x < 0) {`	
`n = -x - 1`	Encode negative numbers with an odd number of bits
`} else {`	
`n = x`	Encode positive numbers with an even number of bits
`}`	
`}`	End of binary alphabet coding
`for(i=0;i<n;i=i+1) {`	Write out a sequence of *n* one-bits
`out(1)`	Write out a single 1-bit
`}`	End of loop writing 1-bits
`out(0)`	Write out 0 as the comma bit
`}`	
`}`	

Annex D
(normative)

Quantization

D.1 General

Inverse quantization computes the wavelet coefficients $c[p,\lambda,b,x]$ in all precincts p, lines λ, bands p and positions x from the decoded quantization index magnitudes $v[p,\lambda,b,x]$ and their signs $s[p,\lambda,b,x]$. Inverse quantization is controlled by the truncation position $T[p,b]$ which depends on the precinct and band, and the bitplane count $M[p,\lambda,b,g]$ of the code group the quantized wavelet coefficient is part of.

This document offers multiple inverse quantization processes, the selection of which is controlled by the *Qpih* elements of the picture header, see subclause A.4.3 for details.

D.2 Inverse deadzone quantization

Table D.1 specifies the inverse deadzone quantization process. The inverse deadzone quantizer is selected if the *Qpih* element of the picture header is 0. The zero bucket of the deadzone quantizer is twice the size of all regular buckets, and the reconstruction point of this quantizer is in the middle of each bucket. The quantization bucket size is given by the truncation position $T[p,b]$ of the precinct p and band b.

Input: Precinct index p, line index λ, band index b, truncation positions $T[p,b]$ of this precinct and band, bitplane counts $M[p,\lambda,b,g]$ of the precinct, line and band and quantization index magnitudes $v[p,\lambda,b,x]$ and their signs $s[p,\lambda,b,x]$.

Output: Wavelet coefficients $c[p,\lambda,b,x]$

Table D.1 — Inverse deadzone quantization

Syntax	Notes
`deadzone_dequant(p,λ,b) {`	
`for(x=0;x<W_pb[p,b];x=x+1) {`	Iterate over all coefficients of band b in precinct p
`g = ⌊x/N_g⌋`	Compute the code group index from the coefficient position
`if (M[p,λ,b,g] > T[p,b] && v[p,λ,b,x] != 0) {`	Check whether a non-zero number of bit-planes is included in this code group and whether the coefficient is non-zero
`r = (1 << T[p,b]) >> 1`	Compute the reconstruction point
`σ = 1-2s[p,λ,b,x]`	Compute the sign of the reconstructed coefficient
`c[p,λ,b,x] = σ × ((v[p,λ,b,x]<< T[p,b]) + r)`	Reconstruct the coefficient
`} else {`	No bitplanes included or coefficient is zero
`c[p,λ,b,x] = 0`	Set to zero
`}`	End of test for sufficient bitplanes and non-zero coefficient
`}`	End of loop over coefficients
`}`	

D.3 Inverse uniform quantization

Table D.2 specifies the inverse uniform quantization process. The inverse uniform quantizer is selected if the *Qpih* element of the picture header is 1. The uniform quantizer uses all equally-sized buckets whose size is determined from the truncation position $T[p,b]$. Compared to the inverse deadzone quantizer, the inverse uniform quantizer requires an additional scaling step.

Input: Precinct index p, line index λ, band index b, truncation positions $T[p,b]$ of this precinct and band, bitplane counts $M[p,\lambda,b,g]$ of the precinct, line and band and quantization index magnitudes $v[p,\lambda,b,x]$ and their signs $s[p,\lambda,b,x]$.

Output: Wavelet coefficients $c[p,\lambda,b,x]$

NOTE The bucket size of the uniform inverse quantizer is given by $\Delta = \frac{2^{M+1}}{2^{M+1-T}-1}$. The reconstruction procedure as given by this subclause is identical to the multiplication of $\sigma \times v[p,\lambda,b,x]$ with Δ within the limits of the implementation precision. This can be seen from the Neumann series $\frac{x}{x-1} = \frac{1}{1-1/x} = \sum_{k=0}^{\infty} x^{-k}$. While multiplication with Δ can also be carried out explicitly, readers should be aware that a single-precision floating point implementation of the above formula will typically generate results different from the algorithm in the following table, and is hence not acceptable.

Table D.2 — Inverse uniform quantization

Syntax	Notes
`uniform_dequant(p,λ,b) {`	
`for(x=0;x<W_pb[p,b];x=x+1) {`	Iterate over all coefficients of band b in precinct p
`g = ⌊x/N_g⌋`	Compute the code group index from the coefficient position
`if (M[p,λ,b,g] > T[p,b] && v[p,λ,b,x] != 0) {`	Check whether a non-zero number of bitplanes is included and whether the coefficient is non-zero
`σ = 1-2s[p,λ,b,x]`	Compute the sign of the reconstructed coefficient
`φ = v[p,λ,b,x]<<T[p,b]`	Get zero-order approximation
`ζ = M[p,λ,b,g] - T[b,p] + 1`	Extract the scale value
`for(ρ = 0; φ > 0; φ = φ >> ζ) {`	Sum over the Neumann series
`ρ = ρ + φ`	Sum up partial terms
`}`	
`c[p,λ,b,x] = σ × ρ`	Insert the sign and reconstruct the coefficient
`} else {`	No bitplanes included or coefficient is zero
`c[p,λ,b,x] = 0`	Set to zero
`}`	End of test for sufficient bitplanes and non-zero
`}`	End of loop over all coefficients
`}`	

D.4 Deadzone quantization

Table D.3 provides guidance on the implementation of a deadzone quantizer whose output is compatible with the normative inverse deadzone quantization procedure specified in subclause D.2.

Input: Precinct index p, line index λ, band index b, truncation positions $T[p,b]$ of this precinct and band, bitplane counts $M[p,\lambda,b,g]$ of the precinct, line and band and wavelet coefficients $c[p,\lambda,b,x]$.

Output: Quantization index magnitudes $v[p,\lambda,b,x]$ and their signs $s[p,\lambda,b,x]$

Table D.3 — Deadzone quantization

Syntax	Notes
`deadzone_quant(p,λ,b) {`	
`  for(x=0;x<W_pb[p,b];x=x+1) {`	Iterate over all coefficients of band b in precinct p
`   if (c[p,λ,b,x] < 0) {`	Test for the sign of the coefficient
`    s[p,λ,b,x] = 1`	Coefficient is negative
`    v[p,λ,b,x] = (-c[p,λ,b,x])>>T[p,b]`	Compute the amplitude from the coefficient
`   } else {`	
`    s[p,λ,b,x] = 0`	Coefficient is positive
`    v[p,λ,b,x] = c[p,λ,b,x]>>T[p,b]`	Compute the amplitude from the coefficient
`   }`	End of the sign check of the coefficient
`  }`	End of loop over all coefficients
`}`	

D.5 Uniform quantization

Table D.4 provides guidance on the implementation of a uniform quantizer whose output is compatible with the normative inverse uniform quantization procedure specified in subclause D.3.

Input: Precinct index p, line index λ, band index b, truncation positions $T[p,b]$ of this precinct and band, bitplane counts $M[p,\lambda,b,g]$ of the precinct, line and band and wavelet coefficients $c[p,\lambda,b,x]$.

Output: Quantization index magnitudes $v[p,\lambda,b,x]$ and and their signs $s[p,\lambda,b,x]$

NOTE The procedure given here is equivalent to mid-point quantization of a scalar quantizer with bucket size $\Delta = \frac{2^{M+1}}{2^{M+1-T}-1}$.

Table D.4 — Uniform quantization

Syntax	Notes
`uniform_quant(p,λ,b) {`	
`  for(x=0;x<W_pb[p,b];x=x+1) {`	Iterate over all coefficients of band b in precinct p
`   g = ⌊x/N_g⌋`	Compute the code group index from the coefficient position
`   if (M[p,λ,b,g] > T[p,b]) {`	Does the coefficient contain sufficient bitplanes?
`    ζ = M[p,λ,b,g] - T[b,p] + 1`	Extract the scale value
`    if (c[p,λ,b,x] < 0) {`	Test for the sign of the coefficient
`     s[p,λ,b,x] = 1`	Coefficient is negative
`     d = -c[p,λ,b,x]`	Compute the amplitude from the coefficient
`    } else {`	
`     s[p,λ,b,x] = 0`	Coefficient is positive
`     d = c[p,λ,b,x]`	Compute the amplitude from the coefficient
`    }`	End of sign check
`    v[p,λ,b,x] = ((d << ζ) - d + (1 << M[p,λ,b,g])) >> (M[p,λ,b,g]+1)`	Quantize and round to nearest

Table D.4 *(continued)*

Syntax	Notes
} else {	Coefficient does not include sufficient number of bitplanes
s[p,b,λ,x] = 0	
v[p,b,λ,x] = 0	Quantize to zero
}	End of check for number of included bitplanes
}	End of loop over all coefficients
}	

D.6 Bitplane count computation

Table D.5 provides guidance on the computation of the bitplane counts $M[p,\lambda,b,g]$ from the wavelet coefficients c[p,λ,b,g]. The bitplane counts $M[p,\lambda,b,g]$ are input to the uniform or deadzone quantization and are encoded in the bitplane count subpacket of the precinct p.

Input: Precinct index p, line index λ, band index b, truncation positions $T[p,b]$ of this precinct and band and wavelet coefficients $c[p,\lambda,b,x]$.

Output: Bitplane counts $M[p,\lambda,b,g]$ of precinct p, line λ and band b.

Table D.5 — Bitplane count computation

Syntax	Notes
compute_bitplane_counts(p,λ,b) {	
for(g=0;g<N_{cg}[b];g=g+1) {	Iterate over all code groups of the band b
v_{max}=0	Set the maximum of the coefficient amplitude to zero
for(k=0;k<N_g;k=k+1) {	Iterate over all members of the code group
x = N_g×g+k	Compute position of the quantized coefficient
if (x < W_{pb}[p,b]) {	Test whether the position is within the band
if (c[p,λ,b,x] < 0) {	Test for the sign of the coefficient
if (-c[p,λ,b,x] >v_{max}) {	Check for a new maximum
v_{max} = -c[p,λ,b,x]	Update the maximum of the coefficient amplitude
}	End of test for a new maximum
} else {	End of test for the sign of the coefficient
if (c[p,λ,b,x] >v_{max}) {	Check for a new maximum
v_{max} = c[p,λ,b,x]	Update the maximum of the coefficient amplitude
}	End of test for a new maximum
}	End of test for the sign of the coefficient
}	End of test whether coefficient is in the band of the precinct
}	End of loop over all code group members
for(m=0;v_{max}>0;v_{max} = v_{max}>>1) {	Loop over bitplanes of v_{max}
m = m+1	Include an additional bitplane
}	End of loop over bitplanes of v_{max}
M[p,λ,b,g] = m	Install bitplane count
}	End of loop over all code groups
}	

Annex E
(normative)

Discrete wavelet transformation

E.1 General

In this annex, the flow charts and tables are normative only in the sense that they are defining an output that alternative implementations shall duplicate.

NOTE In order to achieve a low latency requirement and to conform to one or multiple profiles specified in ISO/IEC 21122-2, decoder implementations would need to run the inverse wavelet transformation steps specified in this annex interleaved with the entropy decoding steps of Annex C and the inverse quantization steps of Annex D. The algorithms given in this annex assume for the ease of presentation that all wavelet coefficients of an image are available entirely.

This annex describes the forward discrete wavelet transformation applied to one component and specifies the inverse discrete wavelet transformation used to reconstruct the component.

E.2 Discrete inverse wavelet transformation

The algorithm specified in Table E.1 takes the wavelet coefficients $c[p,\lambda,b,x]$ of all precincts, lines and bands as input and transforms them by inverse wavelet transformation into the sample values $O[k,x,y]$.

Input: Inversely quantized coefficients $c[p,\lambda,b,x]$ of all precincts, all lines, all bands for all positions.

Output: Inversely wavelet transformed sample values $O[k,x,y]$

Table E.1 — Inverse wavelet transformation

Syntax	Notes
inverse_transformation() {	
for(k=0;k<N_C;k=k+1) {	Loop over components
reorder_coefficients(k)	Rearranges components from all precincts into a rectangular grid
D_x=min(N'$_{L,x}$[k],N'$_{Ly}$[k]))	Compute number of initial horizontal transformations to perform
for(d_x=N'$_{L,x}$[k];d_x>D_x;d_x=d_x−1) {	Loop over horizontal decomposition levels
hor_transform(k,$LL_{dx-1,N'Ly[k]}$,$LL_{dx,N'Ly[k]}$,$HL_{dx,N'Ly[k]}$)	Horizontally transform the $LL_{dx,N'Ly[k]}$ and $HL_{dx,N'Ly[k]}$ bands of component k into the $LL_{dx-1,N'Ly[k]}$ band of component k. The output band is a temporary band that is only required for the inverse wavelet transformation
}	End of the horizontal decompositions
for(d=D_x;d>0;d=d−1) {	Loop over the horizontal and vertical decomposition levels

Table E.1 *(continued)*

Syntax	Notes
hor_transform(k,$LL_{d-1,d}$,$LL_{d,d}$,$HL_{d,d}$)	Horizontally transform the $LL_{d,d}$ and $HL_{d,d}$ bands of component k into the $LL_{d-1,d}$ band of component *k*. The output band is a temporary band that is only required for the inverse computation of the wavelet transformation
hor_transform(k,$LH_{d-1,d}$,$LH_{d,d}$,$HH_{d,d}$)	Horizontally transform the $LH_{d,d}$ and $HH_{d,d}$bands of component *k* into the $LH_{d-1,d}$ band of component *k*. The output band is a temporary band that is only required for the inverse computation of the wavelet transformation.
ver_transform(k,$LL_{d-1,d-1}$,$LL_{d-1,d}$,$LH_{d-1,d}$)	Vertically transform the $LL_{d-1,d}$ band and $LH_{d-1,d}$ band of component *k* into the $LL_{d-1,d-1}$ band of component *k*. The output band is a temporary band that is only required for the inverse computation of the wavelet transformation.
}	End of horizontal and vertical decompositions
assign_output(k,$LL_{0,0}$)	Assign the output of component *k* to the values of the temporary band $LL_{0,0}$
}	End of loop over components
}	

E.3 Coefficient reordering and scaling

The algorithm specified in Table E.2 assigns the dequantized coefficients $c[p,\lambda,b,x]$ from all precincts and component *k* to temporary bands $T[\beta,x,y]$, where β indicates the wavelet filter type and *x* and *y* the sampling position. It also applies an additional scaling step that improves the precision of the wavelet transformation. The temporary bands are required as input to the inverse wavelet filter. The symbols β, $b[\beta.k]$ and N_β are defined in subclause B.3, $b_x[\beta.k]$ in subclause B.4.

Input: Component index *k* and inversely quantized wavelet coefficients $c[p,\lambda,b,x]$ of all precincts, all lines, all bands and all positions. Width $W_b[b]$ and heights $H_b[b]$ of all bands *b*.

Output: Temporary band array $T[\beta,x,y]$ as input to the wavelet filter.

Table E.2 — Coefficient reordering

Syntax	Notes
reorder_coefficients(k) {	
for(β=0;β<N_β;β=β+1) {	Iterate over all subbands of component *k*
if (b_x[β,k]==1) {	Check whether filter type β exists in component *k*
if (k < N_c−Sd) {	Check whether this is a regular component
b=(N_c−Sd)×β+k	Compute the band from the filter type β and the component *k*
} else {	

Table E.2 *(continued)*

Syntax	Notes
`b=(Nc−Sd)×Nβ+k`	Compute the band from the component *k* for non-decomposed components
`}`	End of decision for band computation
`for(y=0;y<Hb[β,k];y=y+1) {`	Iterate over all rows of band *b*
`for(x=0;x<Wb[β,k];x=x+1) {`	Iterate over all colums of band *b*
$p = N_{p,x} \times \left\lfloor \frac{y \times s_y[k] \times 2^{d_y[k,\beta]}}{2^{N_{L,y}}} \right\rfloor + \left\lfloor \frac{x \times s_x[k] \times 2^{d_x[k,\beta]}}{C_s} \right\rfloor$	Compute the precinct index *p* from the horizontal position *x* and vertical position *y*.
$\lambda = y \bmod 2^{N_{L,y} - \log_2 s_y[k] - d_y[k,\beta]}$	Compute the line within the precinct from the vertical position *y*
$\xi = x \bmod \left\lfloor \frac{C_s}{s_x[k] \times 2^{d_x[k,\beta]}} \right\rfloor$	Compute the position within the precinct from the horizontal position *x*
`T[β,x,y]=c[p,λ,b,ξ] << Fq`	Assign and scale the wavelet coefficient in the precinct *p* line *λ* band *b* and horizontal position *ξ* to the temporary band coefficient *T* in band *β*, column *x* and row *y*.
`}`	End of loop over columns
`}`	End of loop over all rows
`}`	End of check over filter existence
`}`	End of loop over all wavelet filter types
`}`	

E.4 Inverse horizontal filtering

The algorithm specified in Table E.3 applies an inverse horizontal wavelet filter on a low-pass and high-pass input band and generates coefficients in a temporary output band.

Input: Component index *k*, output wavelet filter type β_0 and two input filter types, low-pass β_L and high-pass β_H and wavelet coefficients in temporary bands $T[\beta_L,x,y]$ and $T[\beta_H,x,y]$.

Output: Wavelet coefficients in temporary output band $T[\beta_0,x,y]$

Table E.3 — Horizontal inverse wavelet transformation

Syntax	Notes
`hor_transform(k,βo,βL,βH) {`	Horizontally inverse transform the low-pass coefficients β_L and high-pass coefficients β_H to the output band β_o.
`for(y=0;y<Hb[βo,k];y=y+1) {`	Iterate over all rows of band *b*
`for(x=0;x<Wb[βo,k];x=x+1) {`	Iterate over all columns of band *b*
`i = ⌊x/2⌋`	Compute the input sample position in the source band
`if (x umod 2 = 0) {`	If even sample position
`X[x]=T[βL,i,y]`	Assign the low-pass input to the even samples of the temporary array *X*
`} else {`	Else odd sample position
`X[x]=T[βH,i,y]`	Assign the high-pass samples to the odd samples of the temporary array *X*

Table E.3 *(continued)*

Syntax	Notes
}	End of check for even/odd coefficients
}	End of loop over columns
extend_samples(W_b[β_0,k])	Symmetrically extend the samples *X* across the boundary
inverse_filter_1D(W_b[β_0,k])	Performs an inverse filtering on the temporary array *X*
for(x=0;x<W_b[β_0,k];x=x+1) {	Iterate over all columns of band *b*
T[β_0,x,y]=Y[x]	Assign inversely transformed wavelet coefficients to the output band
}	End of loop over columns
}	End of loop over all rows
}	

E.5 Inverse vertical filtering

The algorithm specified in Table E.4 applies an inverse vertical wavelet filter on a low-pass and high-pass input band and generates coefficients in a temporary output band.

Input: Component index *k*, output wavelet filter type β_0 and two input filter types, low-pass β_L and high-pass β_H and wavelet coefficients in temporary bands $T[\beta_L,x,y]$ and $T[\beta_H,x,y]$.

Output: Wavelet coefficients in temporary output band $T[\beta_0,x,y]$

Table E.4 — Vertical inverse wavelet transformation

Syntax	Notes
ver_transform(k,β_o,β_L,β_H) {	Vertically inverse transform the low-pass coefficients β_L and high-pass coefficients β_H to the output band β_o in component *k*.
for(x=0;x<W_b[β_0,k];x=x+1) {	Iterate over all columns of band *b*
for(y=0;y<H_b[β_0,k];y=y+1) {	Iterate over all rows of band *b*
i = ⌊y/2⌋	Compute the input sample position in the source band
if (y umod 2 = 0) {	If even sample position
X[y]=T[β_L,x,i]	Assign the low-pass input to the even samples of the temporary array *X*
} else {	Else odd sample position
X[y]=T[β_H,x,i]	Assign the high-pass samples to the odd samples of the temporary array *X*
}	End of test for even/odd coefficients
}	End of loop over columns
extend_samples(H_b[β_0,k])	Symmetrically extend the samples *X* across the boundary
inverse_filter_1D(H_b[β_0,k]);	Performs an inverse filtering on the temporary array *X*
for(y=0;y<H_b[β_0,k];y=y+1) {	Iterate over all columns of band *b*
T[β_0,x,y]=Y[y]	Assign inversely transformed wavelet coefficients to the output band
}	End of loop over columns
}	End of loop over all rows
}	

E.6 Symmetric extension

The algorithm specified in Table E.5 extends the samples in the temporary array *X* across the boundaries of the band and prepares the sample array *X* for the inverse wavelet transformation.

Input: Array *X*[*x*] of wavelet coefficients and size Z of the array *X*. The array is assumed to be filled for positions 0 to *Z*–1.

Output: Array *X* of wavelet coefficients that have been symmetrically extended.

Table E.5 — Symmetric coefficient extension

Syntax	Notes
`extend_samples(Z) {`	
`  for(i=1;i≤2;i=i+1) {`	Loop over two samples beyond the edge of the temporary array
`   X[-i]=X[i]`	Reflect sample at the left boundary
`   X[Z+i-1]=X[Z-i-1]`	Reflect samples at the right boundary
`  }`	End of loop over sample extension
`}`	

NOTE Due to requirements formulated for the picture header elements W_f, H_f and *Cw*, pathological cases such as empty bands or bands of length *Z*=1 do not appear, such that bands are at least two coefficients wide or high.

E.7 Inverse wavelet filtering with the 5-3 filter

The algorithm specified in Table E.6 computes the inverse wavelet transformation with the 5-3 wavelet filter. It generates from the interleaved low-pass and high-pass input samples *X* output samples in the output array *Y*.

Input: Array *X*[*x*] of wavelet coefficients and size of the array *Z*.

Output: Array *Y*[*x*] of inversely wavelet transformed coefficients, valid at least for the indices 0 to *Z*–1.

Table E.6 — Inverse wavelet filtering with the 5-3 filter

Syntax	Notes
`inverse_filter_1D(Z) {`	
`  for(i=0;i<Z+1;i=i+2) {`	Loop over even samples
`   Y[i]=X[i]-((X[i-1]+X[i+1]+2)>>2)`	Reconstruct even samples from low-pass
`  }`	End of loop over even samples
`  for(i=1;i<Z;i=i+2) {`	Loop over odd samples
`   Y[i]=X[i]+((Y[i-1]+Y[i+1])>>1)`	Reconstruct odd samples from high-pass
`  }`	End of loop over odd samples
`}`	

E.8 Assignment of output coefficients

Table E.7 provides guidance on the assignment of the output of the inverse wavelet transformation contained in the temporary array *T*[*β*,*x*,*y*] to the output array *O*[*k*,*x*,*y*].

Input: Component index *k* and wavelet type *β* indicating an $LL_{0,0}$ band and temporary array of wavelet coefficients *T*[*β*,*x*,*y*] of that band.

Output: Output array $O[c,x,y]$ filled with wavelet coefficients from the $LL_{0,0}$ band of the temporary array $T[\beta,x,y]$.

Table E.7 — Output assignment

Syntax	Notes
`assign_output(k,β) {`	Assign the output of component k from the temporary band β
`for(y=0;y<H_c[k];y=y+1) {`	Loop over the columns of the band data. The height of the component c is denoted by $H_c[k]$ and has been specified in subclause B.1.
`for(x=0;x<W_c[k];x=x+1) {`	Loop over the row of the band data. The width of the component c is denoted by $W_c[k]$ and has been specified in subclause B.1.
`O[k,x,y]=T[β,x,y]`	Assign to the output coefficients O the reconstructed values from the temporary wavelet band $T[\beta,x,y]$
`}`	End of loop over rows
`}`	End of loop over colums
`}`	

E.9 Discrete forwards wavelet transformations

Table E.8 provides guidelines for implementing a forwards wavelet transformation at encoder side. The discrete wavelet transformation takes sample values $O[k,x,y]$ and computes from them wavelet coefficients $c[p,\lambda,b,x]$.

Input: Sample values $O[k,x,y]$ of all components k at all sample positions x and y.

Output: Wavelet coefficients $c[p,\lambda,b,x]$ of all precincts, all lines, all bands for all positions.

Table E.8 provides the steps necessary to implement a forwards wavelet transformation.

Table E.8 — Wavelet transformation

Syntax	Notes
`forwards_transformation() {`	
`for(k=0;k<N_c;k=k+1) {`	Loop over components
`assign_input(k)`	Place data of component k into the input buffer of the wavelet filter
D_x=min(N'$_{L,x}$[k],N'$_{L,y}$[k])	Compute the number of initial transformations to perform
`for(d=1;d≤D_x;d=d+1) {`	Loop over the horizontal and vertical decomposition levels
`ver_fwd_transform(k,LL_{d-1,d-1},LL_{d-1,d},LH_{d-1,d})`	Vertically transform the $LL_{d-1,d-1}$ band into the $LL_{d-1,d}$ and $LH_{d-1,d}$ bands of component k.
`hor_fwd_transform(k,LL_{d-1,d},LL_{d,d},HL_{d,d})`	Horizontally transform the $LL_{d-1,d}$ band into the $LL_{d,d}$ and $HL_{d,d}$ bands of component k.
`hor_fwd_transform(k,LH_{d-1,d},LH_{d,d},HH_{d,d})`	Horizontally transform the $LH_{d-1,d}$ band into the $LH_{d,d}$ and $HH_{d,d}$ bands of component k.
`}`	End of horizontal and vertical decomposition
`for(d_x=D_x+1;d_x≤N'_{L,x}[k];d_x=d_x+1) {`	Loop over horizontal-only decomposition levels

Table E.8 *(continued)*

Syntax	Notes
hor_fwd_transform(k, $LL_{dx-1,N'Ly[k]}$, $LL_{dx,N'Ly[k]}$, $HL_{dx,N'Ly[k]}$)	Horizontally transform the $LL_{dx,N-Ly}$ and $HL_{dx,NLy}$ bands of component k from the $LL_{dx-1,NLy}$ band of component *k*.
}	End of the horizontal decompositions
insert_coefficients(k)	Places coefficients of components into precincts
}	End of loop over components
}	

E.10 Input coefficient assignment

Table E.9 provides guidance on how to assign the sample values in the input array $O[k,x,y]$ to the temporary wavelet band $T[\beta,x,y]$.

Input: Component index *k* and an array of input sample values $O[k,x,y]$.

Output: Temporary array $T[\beta,x,y]$ filled with input data as wavelet coefficients of the $LL_{0,0}$ band.

Table E.9 — Input assignment

Syntax	Notes
assign_input(k) {	Assign the output of component *k* to the temporary $LL_{0,0}$ band
for(y=0;y<H_c[k];y=y+1) {	Loop over the columns of the band data. The height of the component *c* is denoted by $H_b[c]$ and has been specified in subclause B.1.
for(x=0;x<W_c[k];x=x+1) {	Loop over the row of the band data. The width of the component *c* is denoted by $W_c[c]$ and has been specified in subclause B.1.
T[$LL_{0,0}$,x,y]=O[k,x,y]	Assign to the input coefficients *O* the reconstructed values to the temporary wavelet band $T[LL_{0,0},x,y]$, i.e. set the $LL_{0,0}$ band to the input data
}	End of loop over rows
}	End of loop over colums
}	

E.11 Horizontal wavelet transformation

Table E.10 provides guidance on how to perform a horizontal wavelet filter from a temporary input band and to generate low-pass and high-pass output in temporary output bands.

Input: Component index *k*, output wavelet filter type β_0 and two input filter types, low-pass β_L and high-pass β_H; wavelet coefficients in temporary output band $T[\beta_0,x,y]$

Output: Filtered wavelet coefficients in temporary bands $T[\beta_L,x,y]$ and $T[\beta_H,x,y]$.

Table E.10 — Horizontal forward wavelet transformation

Syntax	Notes
hor_fwd_transform(k, β_o, β_L, β_H) {	Horizontally forward transform the low-pass coefficients β_L and high-pass coefficients β_H to the output band β_o in component *k*.
for(y=0;y<H_b[β_0,k];y=y+1) {	Iterate over all rows of band *b*
for(x=0;x<W_b[β_0,k];x=x+1) {	Iterate over all columns of band *b*

Table E.10 *(continued)*

Syntax	Notes
X[x]=T[β_0,x,y]	Copy input coefficients to temporary row
}	End of loop over columns
extend_samples(W$_b$[β_0,k])	Symmetrically extend the samples *X* across the boundary
fwd_filter_1D(W$_b$[β_0,k]);	Performs wavelet filtering on the temporary array *X*
for(x=0;x<W$_b$[b];x=x+1) {	Iterate over all columns of band *b*
i = ⌊x/2⌋	Compute the input sample position in the source band
if (x umod 2 = 0) {	If even sample position
T[β_L,i,y]=Y[x]	Assign the even samples to the low-pass output.
} else {	Else odd sample position
T[β_H,i,y]=Y[x]	Assign the odd samples to the high-pass output.
}	End of even/odd decision
}	End of loop over columns
}	End of loop over all rows
}	

E.12 Vertical wavelet transformation

Table E.11 provides guidance how to perform a vertical wavelet filter from wavelet coefficients in a temporary band and to create low-pass and high-pass output bands.

Input: Component index *k*, output wavelet filter type β_0 and two input filter types, low-pass β_L and high-pass β_H and wavelet coefficients in a temporary input band $T[\beta_0,x,y]$

Output: Wavelet coefficients in temporary output bands $T[\beta_L,x,y]$ and $T[\beta_H,x,y]$.

Table E.11 — Vertical forward wavelet transformation

Syntax	Notes
ver_fwd_transform(k,β_o,β_L,β_H) {	Vertical forward transform the coefficients in the input band β_0 to the low-pass coefficients β_L and high-pass coefficients β_H.
for(x=0;x<W$_b$[β_0,k];x=x+1) {	Iterate over all columns
for(y=0;y<H$_b$[β_0,k];y=y+1) {	Iterate over all rows
X[y]=T[β_0,x,y]	Retrieve one column of the wavelet coefficients and store them in the temporary column
}	End of loop over colums
extend_samples(H$_b$[β_0,k])	Symmetrically extend the samples *X* across the boundary
fwd_filter_1D(H$_b$[β_0,k]);	Performs wavelet filtering on the temporary array *X*
for(y=0;y<H$_b$[β_0,k];y=y+1) {	Iterate over all rows
i = ⌊y/2⌋	Compute the input sample position in the source band
if (y umod 2 = 0) {	If even sample position
T[β_L,x,i]=Y[y]	Assign the even samples to the low-pass output
} else {	Else odd sample position
T[β_H,y,i]=Y[y]	Assign the odd samples to the high-pass output.
}	End of even/odd sample decision
}	End of loop over all columns
}	End of loop over all rows
}	

E.13 Forwards wavelet filtering with the 5-3 filter

Table E.12 provides guidance on how to implement the forwards wavelet transformation with the 5-3 wavelet filter. It generates from the input samples in the array *X* interleaved low-pass and high-pass output in the array *Y*.

Input: Array $X[x]$ of wavelet coefficients and size of the array *Z*.

Output: Array $Y[x]$ of wavelet transformed coefficients, valid at least for the indices 0 to $Z-1$.

Table E.12 — Forward wavelet filtering with the 5-3 filter

Syntax	Notes
`fwd_filter_1D(Z) {`	
`for(i=-1;i<Z+1;i=i+2) {`	Loop over odd samples
`Y[i]=X[i]-((X[i-1]+X[i+1])>>1)`	Generate the high-pass in the odd samples
`}`	End of loop over odd samples
`for(i=0;i<Z;i=i+2) {`	Loop over even samples
`Y[i]=X[i]+((Y[i-1]+Y[i+1]+2)>>2)`	Update the even samples to generate the low-pass
`}`	End of loop over even samples
`}`	

E.14 Insertion of coefficients into precincts

Table E.13 provides guidance on how to assign the wavelet transformed coefficients $c[p,\lambda,b,x]$ in the temporary bands to the precincts.

Input: Component index *k* and temporary band array $T[\beta,x,y]$ containing the output of the wavelet filter step, width $W_b[b]$ and heights $H_b[b]$ of all bands.

Output: Wavelet coefficients $c[p,\lambda,b,x]$ of all precincts, lines, bands and positions.

Table E.13 — Coefficient insertion into precinct

Syntax	Notes
`insert_coefficients(k) {`	
`for(β=0;β<N_β;β=β+1) {`	Iterate over all subbands of component *k*
`if (b_x[β,k] == 1) {`	Check whether the corresponding filter type exists in component *k*
`if (k < N_c-Sd) {`	Check whether this is a regular component
`b=(N_c-Sd)×β+k`	Compute the band from the filter type β and the component *k*
`} else {`	
`b=(N_c-Sd)×N_β+k`	Compute the band from the component *k* for non-decomposed components
`}`	End of decision for band computation
`for(y=0;y<H_b[β,k];y=y+1) {`	Iterate over all rows of band *b*
`for(x=0;x<W_b[β,k];x=x+1) {`	Iterate over all colums of band *b*
$p = N_{p,x} \times \left\lfloor \frac{y \times s_y[k] \times 2^{d_y[k,\beta]}}{2^{N_{L,y}}} \right\rfloor + \left\lfloor \frac{x \times s_x[k] \times 2^{d_x[k,\beta]}}{C_s} \right\rfloor$	Compute the precinct index *p* from the horizontal position x and vertical position *y*.

Table E.13 *(continued)*

Syntax	Notes
$\lambda = y \bmod 2^{N_{L,y} - \log_2 s_y[k] - d_y[k,\beta]}$	Compute the line within the precinct from the vertical position y
$\xi = x \bmod \left\lceil \frac{C_s}{s_x[k] \times 2^{d_x[k,\beta]}} \right\rceil$	Compute the position within the precinct from the horizontal position x
`r = (1<< Fq) >> 1`	Compute the rounding offset
`if (T[β,x,y] >= 0) {`	Check for the sign of the coefficient, if non-negative
`c[p,λ,b,ξ]=(T[β,x,y]+r) >> Fq`	Insert the scaled wavelet coefficient from the temporary band β into precinct p line λ band b and horizontal position x.
`} else {`	If negative
`c[p,λ,b,ξ]=-((-T[β,x,y]+r) >> Fq)`	Insert the scaled wavelet coefficient from the temporary band β into precinct p line λ band b and horizontal position x.
`}`	End of check for sign
`}`	End of loop over columns
`}`	End of loop over all rows
`}`	End of check if band exists
`}`	End of loop over all wavelet filter types
`}`	

Annex F
(normative)

Multiple component transformations

F.1 General

In this annex, the flow charts and tables are normative only in the sense that they are defining an output that alternative implementations shall duplicate.

NOTE In order to achieve a low latency requirement and to conform to one or multiple profiles specified in ISO/IEC 21122-2, decoder implementations would need to run the inverse multiple component steps specified in this annex interleaved with the inverse discrete wavelet transformation steps of Annex F. The algorithms given in this annex assume for the ease of presentation that all sample values of an image are available entirely.

F.2 Inverse multiple component transformation

An inverse multiple component transformation shall be applied to the output sample values $O[c,x,y]$ of the wavelet transformation if the *Cpih* element of the picture header specified in subclause A.4.3 is different from 0, see Table A.9. *Cpih* shall be 0 if there are less than 3 components, i.e. if $N_c<3$, or if any of the sampling factors $s_x[i]$ and $s_y[i]$ for $i<3$ are different from 1. Furthermore, *Cpih* shall be smaller than 2 if there are less than 4 components, i.e. if $N_c<3$, or if any of the sampling factors $s_x[i]$ and $s_y[i]$ for $i<3$ are different from 1.

The inverse multiple component transformation shall be selected from *Cpih* according to Table F.1:

Table F.1 — Selection of the inverse multiple component transformation

Cpih	Inverse Multiple Component Transformation
0	No transformation, set $\Omega[c,x,y] = O[c,x,y]$ for all components c, all columns x and all rows y.
1	inverse_rct(), see subclause F.3, to compute $\Omega[c,x,y]$ from $O[c,x,y]$ for all components $c<3$. Set $\Omega[c,x,y] = O[c,x,y]$ for all components $c\geq 3$.
3	inverse_star_tetrix(), see subclause F.5, to compute $\Omega[c,x,y]$ from $O[c,x,y]$ for all components $c<4$. Set $\Omega[c,x,y] = O[c,x,y]$ for all components $c\geq 4$.
All other values	Reserved for ISO/IEC purposes.

F.3 Inverse reversible multiple component transformation (inverse RCT)

Table F.2 specifies the inverse reversible multiple component transformation that is applied if *Cpih* equals 1:

Input: The output array of the inverse wavelet transformation $O[c,x,y]$ and the dimensions W_f and H_f of the sampling grid.

Output: The intermediate image samples values $\Omega[c,x,y]$ of the image.

Table F.2 — Inverse reversible multiple component transformation

Syntax	Notes
`inverse_rct() {`	
`for(y=0;y<Hf;y=y+1) {`	Loop over rows of the image
`for(x=0;x<Wf;x=x+1) {`	Loop over the columns of the image
i_0 = O[0,x,y]	
i_1 = O[1,x,y]	
i_2 = O[2,x,y]	Retrieve input components
$o_1 = i_0-((i_1+i_2)>>2)$	Reconstruct the green component
$o_0 = o_1 + i_2$	Reconstruct the red component
$o_2 = o_1 + i_1$	Reconstruct the blue component
Ω[0,x,y] = o_0	Assign the red output
Ω[1,x,y] = o_1	Assign the green output
Ω[2,x,y] = o_2	Assign the blue output
}	End of loop over columns
}	End of loop over rows
}	

F.4 Forward reversible multiple component transformation

Table F.3 provides guidance on the forward multiple component colour decorrelation transformation that is inverted by the procedure specified in subclause F.3 at the decoder side.

Input: Scaled intermediate image sample values $\Omega[c,x,y]$ and the dimensions of the sampling grid W_f and H_f.

Output: Decorrelated sample values $O[c,x,y]$ suitable as input of the forward wavelet transformation.

Table F.3 — Forwards reversible multiple component transformation

Syntax	Notes
`forward_rct() {`	
`for(y=0;y<Hf;y=y+1) {`	Loop over rows of the image
`for(x=0;x<Wf;x=x+1) {`	Loop over the columns of the image
i_0 = Ω[0,x,y]	
i_1 = Ω[1,x,y]	
i_2 = Ω[2,x,y]	Retrieve input components
$o_0 = (i_0+2\times i_1+i_2)>>2$	Compute the luma component
$o_1 = i_2 - i_1$	Compute the Cb chroma component
$o_2 = i_0 - i_1$	Compute the Cr chroma component
O[0,x,y] = o_0	Assign the luma output
O[1,x,y] = o_1	Assign Cb
O[2,x,y] = o_2	Assign Cr
}	End of loop over columns
}	End of loop over rows
}	

F.5 Inverse Star-Tetrix transform

F.5.1 Reconstruction with the Star-Tetrix transformation

Table F.4 specifies the inverse Star-Tetrix transformation that is applied if *Cpih* equals 3:

NOTE This algorithm assigns the red decoder output to component 0, the green decoder outputs to components 1 and 2, and the blue component output to component 3, consistent with the rest of this document, in particular with Table F.9.

Input: The output array of the inverse wavelet transformation $O[c,x,y]$ and the dimensions W_f and H_f of the sampling grid.

Output: The intermediate image samples values $\Omega[c,x,y]$ of the image.

Table F.4 — Inverse Star-Tetrix transform

Syntax	Notes	Reference
`inverse_star_tetrix() {`		
`inv_avg_step()`	Reconstruct Y_2 from Y_a and Δ	Table F.5
`inv_delta_step()`	Reconstruct Y_1 from Δ and Y_2	Table F.6
`inv_Y_step()`	Reconstruct G_1 and G_2 from Y_1, Y_2 and C_r, C_b	Table F.7
`inv_CbCr_step()`	Reconstruct R and B from C_r, C_b and G_1, G_2	Table F.8
`for(y=0;y<`H_f`;y=y+1) {`	Loop over rows of the image	
`for(x=0;x<`W_f`;x=x+1) {`	Loop over the columns of the image	
$\Omega[0,x,y] = \omega^4[2,x,y]$	Assign the red output	
$\Omega[1,x,y] = \omega^4[3,x,y]$	Assign the first green output	
$\Omega[2,x,y] = \omega^4[0,x,y]$	Assign the second green output	
$\Omega[3,x,y] = \omega^4[1,x,y]$	Assign the blue output	
`}`	End of loop over columns	
`}`	End of loop over rows	
`}`		

F.5.2 Inverse average step

Table F.5 specifies the inverse average step which reconstructs the Y_2 component from Y_a and Δ. The access() function is specified in subclause F.5.7.

Input: The output array of the inverse wavelet transformation $O[c,x,y]$ and the dimensions W_f and H_f of the sampling grid.

Output: The first lifting step output $\omega^1[c,x,y]$ of the image.

Table F.5 — Inverse average step

Syntax	Notes
`inv_avg_step() {`	
`for(y=0;y<`H_f`;y=y+1) {`	Loop over rows of the image
`for(x=0;x<`W_f`;x=x+1) {`	Loop over the columns of the image
Δ_{lt} `= O[access(0,x,y,-1, -1)]`	Read the delta component to the top-left
Δ_{rt} `= O[access(0,x,y,+1,-1)]`	Read the delta component to the top-right
Δ_{lb} `= O[access(0,x,y,-1,+1)]`	Read the delta component to the bottom-left

Table F.5 *(continued)*

Syntax	Notes
Δ_{rb} = O[access(0,x,y,+1,+1)]	Read the delta component to the bottom-right
$\omega^1[0,x,y]=O[0,x,y]-\left\lfloor\frac{\Delta_{lt}+\Delta_{rt}+\Delta_{lb}+\Delta_{rb}}{8}\right\rfloor$	Reconstruct Y_2 from Y_a and Δ
$\omega^1[1,x,y]=O[1,x,y]$	Copy remaining components over
$\omega^1[2,x,y]=O[2,x,y]$	Copy remaining components over
$\omega^1[3,x,y]=O[3,x,y]$	Copy remaining components over
}	End of loop over columns
}	End of loop over rows
}	

F.5.3 Inverse delta step

Table F.5 specifies the inverse delta step which reconstructs the Y_1 component from Y_2 and Δ. The access() function is specified in subclause F.5.7.

Input: The output array of the inverse average step $\omega^1[c,x,y]$ and the dimensions W_f and H_f of the sampling grid.

Output: The second lifting step output $\omega^2[c,x,y]$ of the image.

Table F.6 — Inverse delta step

Syntax	Notes
inv_delta_step() {	
for(y=0;y<H_f;y=y+1) {	Loop over rows of the image
for(x=0;x<W_f;x=x+1) {	Loop over the columns of the image
Y_{lt} = ω^1[access(3,x,y,-1,-1)]	Read the Y_2 component to the top-left
Y_{rt} = ω^1[access(3,x,y,+1,-1)]	Read the Y_2 component to the top-right
Y_{lb} = ω^1[access(3,x,y,-1,+1)]	Read the Y_2 component to the bottom-left
Y_{rb} = ω^1[access(3,x,y,+1,+1)]	Read the Y_2 component to the bottom-right
$\omega^2[3,x,y]=\omega^1[3,x,y]+\left\lfloor\frac{Y_{lt}+Y_{rt}+Y_{lb}+Y_{rb}}{4}\right\rfloor$	Reconstruct Y_1 from Y_2 and Δ
$\omega^2[1,x,y]=\omega^1[1,x,y]$	Copy remaining components over
$\omega^2[2,x,y]=\omega^1[2,x,y]$	Copy remaining components over
$\omega^2[0,x,y]=\omega^1[0,x,y]$	Copy remaining components over
}	End of loop over columns
}	End of loop over rows
}	

F.5.4 Inverse Y step

Table F.7 specifies the inverse Y step which reconstructs the G_1 and G_2 components from Y_1, Y_2 and C_b, C_r. The access() function is specified in subclause F.5.7.

Input: The output array of the second lifting step $\omega^2[c,x,y]$, the chroma weighting exponents e_1 and e_2 and the dimensions W_f and H_f of the sampling grid.

Output: The third lifting step output $\omega^3[c,x,y]$ of the image.

Table F.7 — Inverse Y step

Syntax	Notes
inv_Y_step() {	
for(y=0;y<H_f;y=y+1) {	Loop over rows of the image
for(x=0;x<W_f;x=x+1) {	Loop over the columns of the image
B_l = ω^2[access(0,x,y,−1,0)]	Read the C_b component to the left of G_2
B_r = ω^2[access(0,x,y,+1,0)]	Read the C_b component to the right of G_2
R_t = ω^2[access(0,x,y,0,−1)]	Read the C_r component to the top of G_2
R_b = ω^2[access(0,x,y,0,+1)]	Read the C_r component to the bottom of G_2
$\omega^3[0,x,y]=\omega^2[0,x,y]-\left\lfloor\frac{2^{e_2}(B_l+B_r)+2^{e_1}(R_t+R_b)}{8}\right\rfloor$	Compute G_2 from Y_2 and neighbouring C_b and C_r samples
B_t= ω^2[access(3,x,y,0,−1)]	Read the C_b component to the top of G_1
B_b= ω^2[access(3,x,y,0,+1)]	Read the C_b component to the bottom of G_1
R_l = ω^2[access(3,x,y,−1,0)]	Read the C_r component to the left of G_1
R_r = ω^2[access(3,x,y,+1,0)]	Read the C_r component to the right of G_1
$\omega^3[3,x,y]=\omega^2[3,x,y]-\left\lfloor\frac{2^{e_2}(B_t+B_b)+2^{e_1}(R_l+R_r)}{8}\right\rfloor$	Compute G_1 from Y_1 and neighbouring C_b and C_r samples
ω^3[1,x,y]=ω^2[1,x,y]	Copy remaining components over
ω^3[2,x,y]=ω^2[2,x,y]	Copy remaining components over
}	End of loop over columns
}	End of loop over rows
}	

F.5.5 Inverse C_bC_r step

Table F.8 specifies the inverse C_bC_r step which reconstructs the R and B components from G_1 and G_2 and C_b, C_r. The access() function is specified in subclause F.5.7.

Input: The output array of the third lifting step $\omega^3[c,x,y]$ and the dimensions W_f and H_f of the sampling grid.

Output: The fourth lifting step output $\omega^4[c,x,y]$ of the image.

Table F.8 — Inverse C_bC_r step

Syntax	Notes
inv_cbcr_step() {	
for(y=0;y<H_f;y=y+1) {	Loop over rows of the image
for(x=0;x<W_f;x=x+1) {	Loop over the columns of the image
G_l = ω^3[access(1,x,y,−1,0)]	Read the G component to the left of C_b
G_r = ω^3[access(1,x,y,+1,0)]	Read the G component to the right of C_b
G_t = ω^3[access(1,x,y,0,−1)]	Read the G component to the top of C_b
G_b = ω^3[access(1,x,y,0,+1)]	Read the G component to the bottom of C_b
$\omega^4[1,x,y]=\omega^3[1,x,y]+\left\lfloor\frac{G_l+G_r+G_t+G_b}{4}\right\rfloor$	Compute B from C_b and neighbouring G samples
G_l = ω^3[access(2,x,y,−1,0)]	Read the G component to the left of C_r

Table F.8 *(continued)*

Syntax	Notes
G_r = ω³[access(2,x,y,+1,0)]	Read the G component to the right of C_r
G_t = ω³[access(2,x,y,0,-1)]	Read the G component to the top of C_r
G_b = ω³[access(2,x,y,0,+1)]	Read the G component to the bottom of C_r
$\omega^4[2,x,y]=\omega^3[2,x,y]+\left\lfloor\frac{G_l + G_r + G_t + G_b}{4}\right\rfloor$	Compute R from C_r and neighbouring G samples
ω⁴[0,x,y]=ω³[0,x,y]	Copy remaining components over
ω⁴[3,x,y]=ω³[3,x,y]	Copy remaining components over
}	End of loop over columns
}	End of loop over rows
}	

F.5.6 Super pixel look-up tables

Table F.9 specifies the CFA pattern type C_t depending on the values of the component registration values *Xcrg*[*c*],*YCrg*[*c*] found in the component registration marker, see subclause A.4.9.

Table F.9 — CFA Pattern type derived from the component registration

Component index *c*	Xcrg[c]	Ycrg[c]	CFA Pattern Type C_t	Notes (Subpixel arrangement and chroma exponent assignment)
0	0	0	0	RGGB e_1 is the C_r weight e_2 is the C_b weight
1	32768	0		
2	0	32768		
3	32768	32768		
0	32768	32768	0	BGGR e_1 is the C_b weight e_2 is the C_r weight
1	32768	0		
2	0	32768		
3	0	0		
0	32768	0	1	GRBG e_1 is the C_r weight e_2 is the C_b weight
1	0	0		
2	32768	32768		
3	0	32768		
0	0	32768	1	GBRG e_1 is the C_b weight e_2 is the C_r weight
1	0	0		
2	32768	32768		
3	32768	0		
All other combinations of *Xcrg*[] and *YCrg*[]			Reserved for ISO/IEC purposes	

NOTE Exchanging the registration of R and B does not change C_t, but can require exchanging e_1 and e_2 in the CTS marker segment as indicated in the table.

Table F.10 specifies for each component index *c* between 0 the displacement vector δ_x,δ_y within a CFA super pixel, dependent on the CFA pattern type C_t. The value of C_t is defined in Table F.9.

Table F.10 — Component displacement vector by component index

Component index c	CFA pattern type C_t	Displacement vector δ_x,δ_y
0	0	0,1
1	0	1,1
2	0	0,0
3	0	1,0
0	1	1,1
1	1	0,1
2	1	1,0
3	1	0,0

Table F.11 specifies its inverse function $k[\delta_x,\delta_y]$. Given a displacement vector δ_x,δ_y and a CFA pattern type C_t, $k[\delta_x,\delta_y]$ evaluates to the component index c at the given displacement vector within a CFA super pixel.

Table F.11 — Component index by displacement vector

Displacement vector δ_x,δ_y	CFA pattern type C_t	Component index $k[\delta_x,\delta_y]$
0,1	0	0
1,1	0	1
0,0	0	2
1,0	0	3
1,1	1	0
0,1	1	1
1,0	1	2
0,0	1	3

F.5.7 Coordinate access function

Table F.12 specifies a function that computes the component and coordinates of a sub-pixel relative to a sub-pixel at a given coordinate and of a given component

Input: The coordinates (x,y) and component c in the sample grid of a super pixel of a CFA array, a sample offset r_x and r_y the sample grid dimensions W_f and H_f, the colour transformation CFA pattern type C_t and the colour transformation reflection and extension flags C_f.

Output: A triple (c,x,y) of component c, x position and y position within the sample grid.

Table F.12 — Coordinate access function

Syntax	Notes
access(c,x,y,r_x,r_y){	
if ((2x+r_x+δ_x[c] < 0) || (2x+r_x+δ_x[c] >= 2W_f)) {	Check whether the access would go beyond the sample grid
r_x =−r_x	If so, reflect back into the line
}	
if ((C_f == 3 && r_y+δ_y[c] < 0)|| (C_f == 3 && r_y+δ_y[c] > 1)|| (2y+r_y+δ_y[c] < 0) || (2y+r_y+δ_y[c] >= 2H_f)) {	Check whether the access is in-line and an access is attempted to the line above, or the access is in-line, and an access is made to the line below, or the access would go beyond the sample grid
r_y =−r_y	If so, reflect back into the line