INTERNATIONAL STANDARD

ISO/IEC 21122-1

First edition
2019-05

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

Part 1:
Core coding system

Reference number
ISO/IEC 21122-1:2019(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
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).

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

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

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

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

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

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

Introduction

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

ISO and IEC take no position concerning the evidence, validity and scope of this patent right. The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from:

INTOPIX SA

Rue Emile Francqui 9

B-1435 Mont-Saint-Guibert, Belgium

Fraunhofer-Gesellschaft zur Foerderung der angewandten Forschung e.V. for its Fraunhofer Institute for Integrated Circuits IIS

Am Wolfsmantel 33

91058 Erlangen, Germany

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. 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 a 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 a decoding process 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 terminological databases for use in standardization at the following addresses:

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

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

3.1.1
band
input data to a specific *wavelet filter type* (3.1.49) that contributes to the generation of one of the *components* (3.1.13) of the image

3.1.2
band type
single number collapsing the information on the *component* (3.1.13), and horizontal and vertical *wavelet filter types* (3.1.49) that are applied in the filter cascade reconstructing spatial image *samples* (3.1.42) from inversely quantized wavelet *coefficients* (3.1.10)

3.1.3
bit
binary choice encoded as either 0 or 1

3.1.4
bitplane
array of *bits* (3.1.3) having all the same *significance* (3.1.31)

3.1.5
bitplane count
number of significant *bitplanes* (3.1.4) of a *code group* (3.1.9), counting from the LSB up to the most significant, non-empty bitplane

3.1.6
bitplane count subpacket
subset of a *packet* (3.1.34) which decodes to the *bitplane counts* (3.1.5) of all *code groups* (3.1.9) within a packet, followed by *padding* (3.1.35) and optional *filler bytes* (3.1.24)

Note 1 to entry: See subclause C.5.3.

3.1.7
byte
group of 8 *bits* (3.1.3)

3.1.8
codestream
compressed image data representation that includes all necessary data to allow a (full or approximate) reconstruction of the *sample* (3.1.42) values of a digital image

3.1.9
code group
group of *quantization indices* (3.1.40) in sign-magnitude representation before *inverse quantization* (3.1.25)

3.1.10
coefficient
input value to the inverse wavelet transformation resulting from *inverse quantization* (3.1.25)

3.1.11
column
set of vertically aligned *precincts* (3.1.36)

3.1.12
compression
process of reducing the number of *bits* (3.1.3) used to represent source image data

3.1.13
component
two-dimensional array of *samples* (3.1.42) having the same designation such as red, green or blue in the output or display device

3.1.14
continuous-tone image
image whose *components* (3.1.13) have more than one *bit* (3.1.3) per *sample* (3.1.42)

3.1.15
data subpacket
subset of a *packet* (3.1.34) which consists of the *quantization index magnitudes* (3.1.41), followed by *padding* (3.1.35) and optional *filler bytes* (3.1.24)

Note 1 to entry: See subclause C.5.4.

3.1.16
deadzone quantizer
quantizer whose zero bucket has a size different from all other buckets

Note 1 to entry: Based on this, inverse deadzone quantizers can be defined as inverse quantizers whose zero bucket has a size different from all other buckets.

3.1.17
decoder
embodiment of a *decoding process* (3.1.18)

3.1.18
decoding process
process which takes as its input a *codestream* (3.1.8) and outputs a *continuous-tone image* (3.1.14)

3.1.19
decomposition level
set of wavelet *coefficients* (3.1.10) resulting from a particular level of recursive application of a wavelet transform

3.1.20
encoder
embodiment of an *encoding process* (3.1.23)

3.1.21
encoding process
process which outputs compressed image data in the form of a *codestream* (3.1.8)

3.1.22
entropy decoding
lossless (3.1.28) *procedure* (3.1.38) which recovers the sequence of symbols from the sequence of *bits* (3.1.3) produced by an *entropy encoding* (3.1.23) procedure

3.1.23
entropy encoding
lossless (3.1.28) *procedure* (3.1.38) which converts a sequence of input symbols into a sequence of *bits* (3.1.3) such that the average number of bits per symbol approaches the entropy of the input symbols

3.1.24
filler bytes
integer number of *bytes* (3.1.7) a *decoder* (3.1.17) will skip over on decoding without interpreting the values of the bytes itself

3.1.25
inverse quantization
inverse *procedure* (3.1.38) to *quantization* (3.1.39) by which the *decoder* (3.1.17) recovers a representation of the *coefficients* (3.1.10)

3.1.26
inverse reversible multi component transformation
inverse RCT
inverse transformation across multiple *component* (3.1.13) *sample* (3.1.42) values located at the same *sample grid* (3.1.43) point that is invertible without loss

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

3.1.27
LL band
input to a series of wavelet filters where only inverse low-pass filters are applied in horizontal and vertical direction

3.1.28
lossless
being such that, for encoding and decoding *procedures* (3.1.38), the output of the decoding procedure(s) is identical to the input to the encoding procedure(s)

3.1.29
lossless coding
mode of operation which refers to any one of the coding processes defined in this document in which all of the *procedures* (3.1.38) are *lossless* (3.1.28)

3.1.30
sign subpacket
subset of a *packet* (3.1.34) that consists of the sign information of all non-zero *quantization indices* (3.1.40) within a packet, followed by *padding* (3.1.35) and optional *filler bytes* (3.1.24)

Note 1 to entry: See subclause C.5.5.

3.1.31
significance
attribute of *code groups* (3.1.9) that applies if, depending on the Run Mode flag in the picture header, either at least one of *coefficients* (3.1.10) in the code group is non-zero, or the *bitplane count* (3.1.5) prediction residual of the code group is non-zero

3.1.32
significance group
group of horizontally adjacent *code groups* (3.1.9) sharing the same *significance* (3.1.31) information in the *significance subpacket* (3.1.33)

3.1.33
significance subpacket
subset of a *packet* (3.1.34) that identifies which *significance groups* (3.1.32) within a packet are insignificant, followed by *padding* (3.1.35) and optional *filler bytes* (3.1.24)

Note 1 to entry: See subclause C.5.2.

3.1.34
packet
segment of the *codestream* (3.1.8) containing entropy coded information on a single *precinct* (3.1.36), line and a subset of the *bands* (3.1.1) within this precinct and line

3.1.35
padding
bits (3.1.3) within the *codestream* (3.1.8) whose only purpose is to align syntax elements to *byte* (3.1.7) boundaries and that carry no information

3.1.36
precinct
collection of *quantization indices* (3.1.40) of all *bands* (3.1.1) contributing to a given spatial region of the image

3.1.37
precision
number of *bits* (3.1.3) allocated to a particular *sample* (3.1.42), *coefficient* (3.1.10), or other binary numerical representation

3.1.38
procedure
set of steps which accomplishes one of the tasks which comprise an *encoding* (3.1.23) or *decoding process* (3.1.18)

3.1.39
quantization
method of reducing the *precision* (3.1.37) of the individual *coefficients* (3.1.10)

3.1.40
quantization index
input to the *inverse quantization* (3.1.25) process which reconstructs a wavelet *coefficient* (3.1.10)

3.1.41
quantization index magnitude
absolute value of a *quantization index* (3.1.40)

3.1.42
sample
single element in the two-dimensional image array which comprises a *component* (3.1.13)

3.1.43
sample grid
common coordinate system for all *samples* (3.1.42) of an image, where 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

3.1.44
slice
integral number of *precincts* (3.1.36) whose wavelet *coefficients* (3.1.10) can be entropy-decoded independently

3.1.45
subpacket
substructure of a *packet* (3.1.34) containing information of one or multiple *bands* (3.1.1) of one line of a single *precinct* (3.1.36)

3.1.46
truncation position
number of least significant *bitplanes* (3.1.4) not included in the *quantization index* (3.1.40) of a wavelet *coefficient* (3.1.10)

3.1.47
uniform quantizer
quantizer whose buckets are all of equal size

Note 1 to entry: Based in this, inverse uniform quantizers can be defined as inverse quantizers whose buckets are all of equal size.

3.1.48
upsampling
procedure (3.1.38) by which the spatial resolution of a *component* (3.1.13) is increased

3.1.49
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

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
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,λ,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
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
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[a]	height of subband a 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,λ,s]	line inclusion flag, set if line λ of band b and precinct p is included in packet s, reset otherwise
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
Ldat[p,s]	size of the data subpacket of precinct p and packet s in bytes
Lcnt[p,s]	size of the bitplane count subpacket of precinct p and packet s in bytes
Lsgn[p,s]	size of the sign subpacket of precinct p and packet s in bytes
Lprc[p]	length of the entropy coded data in precinct p
Lslc	slice length 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)
$N_{L,x}$	number of horizontal decomposition levels
$N_{L,y}$	number of vertical decomposition levels
$N_{p,x}$	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
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
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
$T[p,b]$	truncation position of precinct p and band b

$T_{top}[p,b]$	vertical truncation position predictor of precinct p and band b
T[β,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,λ,b,x]	quantization index magnitude of the wavelet coefficient in precinct p, line λ, band b and position x
$W_b[b]$	width of band b 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
Yslh	vertical slice order within the picture
Z[p,λ,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

+	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
Umod	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

|x| absolute value of x, |x| equals –x for x < 0, otherwise x

sign(x)	sign of x, 0 if x is 0, +1 if x is positive, -1 if x is negative
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

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. It neither specifies the registration of the components relative to each other. The sampling grid provides only an abstract coordinate system for the computation of positions and dimensions of codestream elements.

5.2 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. However, 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.

5.3 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. Bits within bytes are transmitted in decreasing magnitude order, with the MSB of a byte transmitted first and the LSB transmitted last.

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 document 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 1 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.

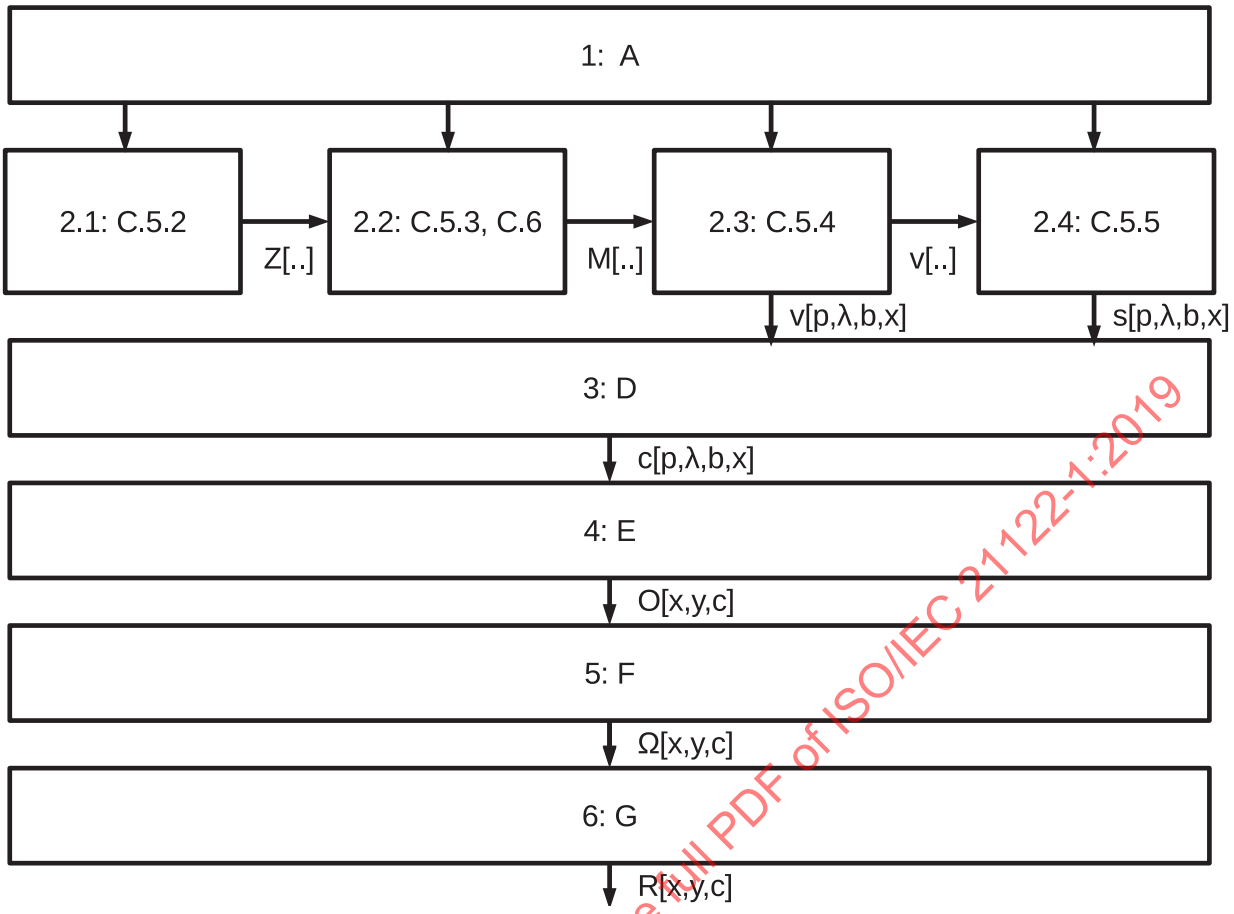


Figure 1 — Decoder overview

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 1.

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,λ,b,j]**, significance information indicates the presence of significant code groups within the $\mathbf{j}^{\text{th}}$ 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 C.6. The integer array **M[p,λ,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,λ,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,λ,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,λ,b,x]** and signs **s[p,λ,b,x]** are then inversely quantized by the dequantizer specified in Annex D, giving wavelet coefficients **c[p,λ,b,x]**.

In block 4, wavelet coefficients **c[p,λ,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 **Ω[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 **Ω[x,y,c]**, 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 $\mathbf{x \times s_x[c]}$, $\mathbf{y \times s_y[c]}$. 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, in the sense that the decoded output image produced by an implementation of a decoder shall be the same as the decoded output image produced by the decoding processes specified in this document, including those processes specified in Annexes A through G.

There is no normative or required specification for the particular internal steps or ordering of internal operations to be performed within the decoder that are 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. 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 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 of 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 may 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 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 1 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.5
picture_header()	Defines the overall structure of the codestream	Table A.6
component_table()	Defines the precision and sampling factors of all components in the image	Table A.13
weights_table()	Defines weight and gain factors that steer the decoding process	Table A.16
extension_marker()	Optional extension of the codestream syntax	Table A.14
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.17
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	

Table A.1 *(continued)*

Syntax	Notes	Defined in
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
}		
}		
EOC_marker()	Identifies the end of the JPEG XS codestream	Table A.4
}		

NOTE 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 Future extensions of this document can 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, it is advisable that lower level transport mechanisms regain synchronization as it is not possible to search the codestream for markers; indeed, 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.4.7
0xff15	COM	Extension marker	Optional	A.4.6
0xff20	SLH	Slice header	Mandatory	A.4.8
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 behind the SOC marker.

Currently, this document does not define any capabilities, hence Lcap=2 and cap[i]=0 for all i. Future extensions will be signalled by a non-zero element in the cap[i] array. Nevertheless, the CAP marker shall be present.

Table A.5 — 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)	
}			
Padding	Pad to an integer number of bytes	pad(8)	
}			

A.4.4 Picture header

Table A.6 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.6 — 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)	$\max_i\left(s_x[i]\right)\times 2^{N_{L,x}}$ — (2^{16} – 1)
$\mathbf{H_f}$	Height of the image in sample grid positions	u(16)	$\max_i\left(s_y[i]\right)\times 2^{N_{L,y}}$ — (2^{16} – 1)

Table A.6 *(continued)*

Syntax	Notes	Size	Values
Cw	Width of a precinct in multiples of $8\times\max_i\left(s_x[i]\right)\times 2^{N_{L,x}}$ sample positions, other than the rightmost precincts. If this field is 0, precincts are as wide as the image. See subclause B.4 for details.	u(16)	0 — $(2^{16}-1)$ such that either Cw=0 or $W_f \bmod (8\times C_w \times \max_i\left(s_x[i]\right)\times 2^{N_{L,x}}) \geq \max_i(s_x[i])\times 2^{N_{L,x}}$
H_{sl}	Height of a slice in precincts other than the last slice	u(16)	1 — $(2^{16}-1)$
N_c	Number of components in the image	u(8)	1 — 8
N_g	Number of coefficients per code group	u(8)	4
S_s	Number of code groups per significance group	u(8)	8
Bw	Nominal bit precision of the wavelet coefficients	u(8)	20
Fq	Number of fractional bits in the wavelet coefficients	u(4)	8
B_r	Number of bits to encode a bitplane count in raw	u(4)	4
Fslc	Slice coding mode	u(1)	See Table A.12
Ppoc	Progression order of bands within precincts	u(3)	See Table A.11
Cpih	Colour transformation to be used for inverse decorrelation	u(4)	See Table A.7
$N_{L,x}$	Number of horizontal wavelet transformations	u(4)	1 — 8
$N_{L,y}$	Number of vertical wavelet transformations	u(4)	0 — min($N_{L,x}$,6)
Qpih	Inverse quantizer type	u(4)	See Table A.8
Fs	Sign handling strategy	u(2)	See Table A.9
Rm	Run mode	u(2)	See Table A.10
}			

NOTE 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.

Table A.7 — Colour transformation

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

Table A.8 — Quantizer type

Qpih	Meaning
0	Deadzone quantizer (see Annex D)
1	Uniform quantizer (see Annex D)
2–15	Reserved for ISO/IEC purposes

Table A.9 — 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.10 — Run mode

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

Table A.11 — Progression order

Ppoc	Meaning
0	The progression order as defined by subclause B.6 (resolution-line-band-component)
1–7	Reserved for ISO/IEC purposes

Table A.12 — 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.13 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.13 — 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<N_c;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 for all components, 1 or 2 for components c > 0
s_y[c]	Vertical sampling factor of component c	u(4)	1
}	End of loop over components		
}			

A.4.6 Extension marker

Table A.14 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.14 — 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.15
Dcom	User-defined data	Variable	Variable
Padding	Padding to an integer number of bytes	pad(8)	0
}			

Table A.15 — Tcom encoding

Tcom	Meaning
0x0000	Vendor of the encoder, Dcom is a zero-terminated, ISO/IEC 646 encoded string identifying the vendor of the encoder
0x0001	Copyright statement that the codestream creator wants to convey to users of the codestream. The interpretation of this statement is beyond the scope of this document. Dcom is a zero-terminated, ISO/IEC 646 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.4.7 Weights table

Table A.16 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.

Table A.16 — 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)	$2 \times N_L + 2$
for(b=0;b < N_L; b = b+1) {	Loop over all bands.		
G[b]	Gain of band b	u(8)	0–15
P[b]	Priority of band b	u(8)	0-255
}	End of loop over bands		
}			

NOTE Annex H lists example configurations for the weights table specified in Table A.16. 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.4.8 Slice header

Table A.17 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.17 — 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.

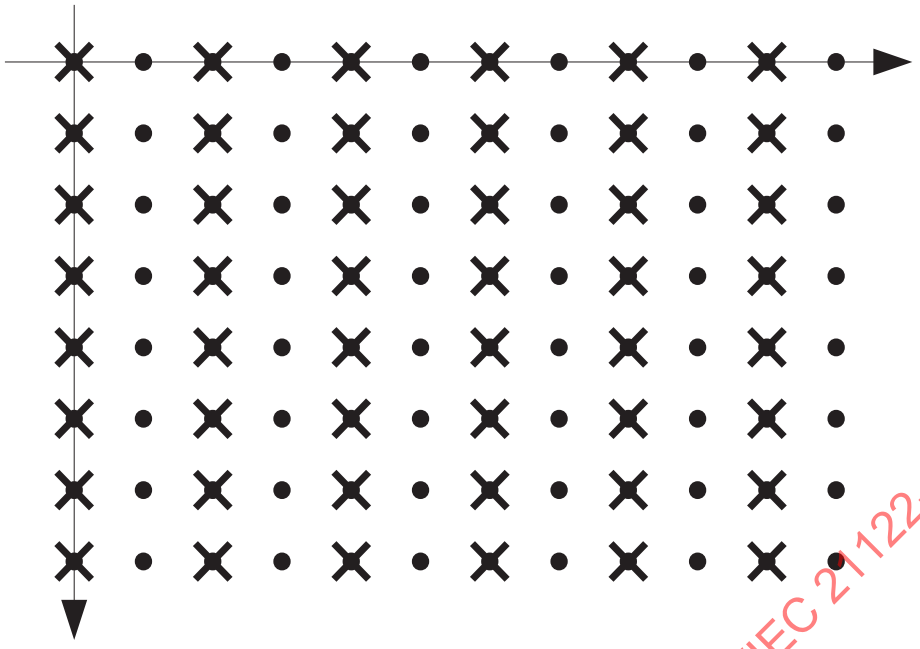
The number of samples in every row of component **i** is given by

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

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.



NOTE 1 In this figure, $s_x[0]=s_y[0]=1$ indicated by round dots, and $s_x[i]=2$ $s_y[i]=1$ for all other components indicated by crosses, 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 sampled image or how components are registered relative to each other; such information is beyond the scope of this document. In particular, both centered and co-sited positioning of subsampled components are possible, though no syntax elements are provided within this document to distinguish these cases.

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

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

B.2 Division of the subsampled image plane into bands

Each of the components is decomposed by a wavelet transformation with $N_{L,x}$ horizontal and $N_{L,y}$ vertical decomposition levels. 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. Finally, the filter type **β** and the component index **c** are combined into a band index **b** by a procedure also given in subclause B.3.

The width $W_b[b]$ of band **b** at $d_x[\beta]$ horizontal decompositions of component **i** is given by:

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

The height $H_b[b]$ of band **b** at $d_y[\beta]$ vertical decomposition levels of component **i** is given by:

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

In case of 0 vertical wavelet decompositions, a vertical high-pass does not exist and $H_b[b]$ is identical to $H_c[i]$ for all bands **b** 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 $N_\beta-1$, 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 is not 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.

Table B.1 provides the resulting enumeration of wavelet filter types for 0 vertical and 5 horizontal wavelet decompositions, Table B.2 the resulting enumeration of wavelet filter types for 1 vertical and 5 horizontal wavelet decompositions and Table B.3 the resulting enumeration of wavelet filter types for 2 vertical and 5 horizontal wavelet decompositions.

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:

$$N_L=N_c\times N_\beta$$

The band index **b** is computed from the wavelet filter type **β** and the component index **i** in the following way:

$$b=N_c\times\beta+i$$

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	$HL_{4,0}$

Table B.1 *(continued)*

Wavelet filter type β	Wavelet filtering and decomposition depths (subscripts are d_x and d_y)
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 3 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.

B.4 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:

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

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**, i.e. x_b runs from 0 to $W_b[b]-1$ and y_b runs from 0 to $H_b[b]-1$, with $W_b[b]$ and $H_b[b]$ the band dimensions as defined in subclause B.2. Then a sample in band **b** at position (x_b, y_b) is part of precinct **p** if and only if

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

where $d_x[\beta]$ and $d_y[\beta]$ is the horizontal respectively vertical decomposition depth of the filter type **β** of band **b.** The filter type **β** and the component **i** are computed from the band **b** as follows:

$$\beta = \lfloor b / N_c \rfloor \qquad i = b \bmod N_c$$

$N_{L,x}$ 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 $W_{pb}[p,b] = \left\lceil \frac{W_p[p]}{2^{d_x[\beta]}} \right\rceil$ for a horizontally low-pass filtered band and $W_{pb}[p,b] = \left\lfloor \left\lceil \frac{W_p[p]}{2^{d_x[\beta]-1}} \right\rceil / 2 \right\rfloor$ 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.5 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:

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

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:

$$L_0\left[b,p\right]=2^{N_{L,y}-d_y[\beta]}\cdot\tau_y\left[b\right] \text{ and}$$

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

where β is computed from b as indicated in subclause B.4. Figure B.2 provides an example how an image is divided into precincts, lines, bands and packets.

NOTE 1 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.

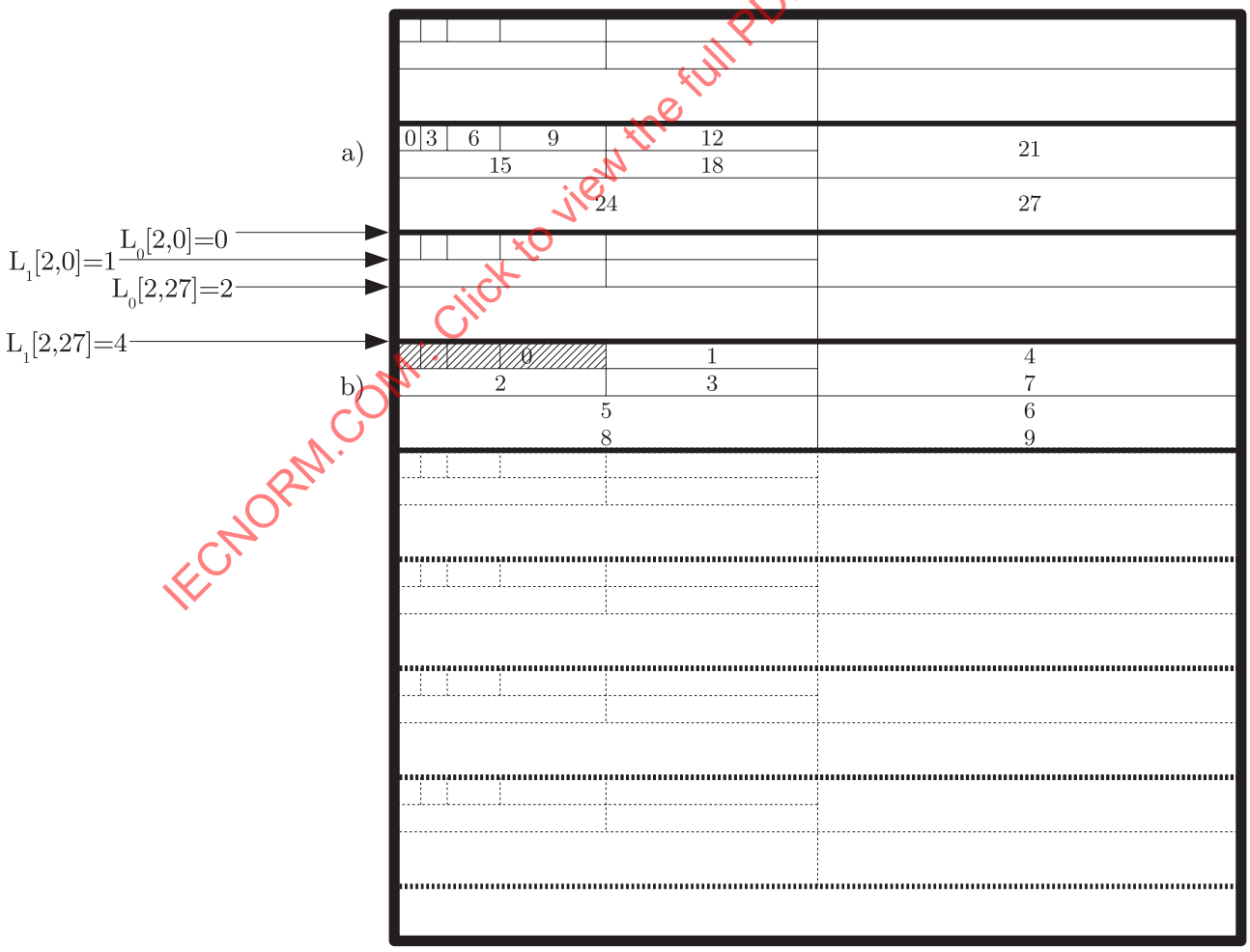


Figure B.2 — Precincts, slices and packets

NOTE 2 In Figure B.2, a 5 level horizontal and 2 level vertical decomposition with a slice-height of 4 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. 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. The presence of a band in the precinct for the last precinct of the picture cannot be easily inferred from Figure B.2.

B.6 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,λ,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,λ,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,λ,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) {	
s=0	Reset packet index
$\beta_1=\max(N_{L,x},N_{L,y})-\min(N_{L,x},N_{L,y})+1$	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;i=i+1) {	Loop over components
I(p,N_c×β+i,0,s)=1	β_1 bands of all components included in the first packet
}	
}	
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<$2^{N_{L,y}-d_y[\beta_0]}$;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
if (λ+L_0[p, N_c×β] < L_1[p, N_c×β]) {	Check whether the line is in the precinct
s=s+1	Create the next packet
for(i=0;i<N_c;i=i+1) {	Loop over components
I(p,N_c×β+i,λ+L_0[p, N_c×β],s)=1	
}	End of loop over components
}	End of test for line within the precinct
}	End of loop over filter types
}	End of loop over lines

Table B.4 *(continued)*

Name	Notes
}	End of loop over proxy levels
$N_{pc}[p]=s+1$	Define the number of packets in total
}	

NOTE The above algorithm results for 3 components, 5 horizontal and 0 vertical wavelet decomposition 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.

Table B.5 — Line inclusion flags for zero vertical decomposition level

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

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

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)

B.7 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:

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

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.8 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 **p**. 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:

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

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.9 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** consists 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 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.9) 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.4.8 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,λ,b,x]** and wavelet coefficient signs **s[p,λ,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 it contains only zero coefficients, or its bitplane count 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 $\mathbf{D_r[p,s]}$ 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,λ,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,λ,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,λ,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,λ,b,x]** the sign **s[p,λ,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 Lprc[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.

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

Output: size of the precinct in bytes including filler bytes **Lprc[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) {			
Lprc[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		
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 $\mathbf{D_r[p,s]}$ field in the subpacket header; see Table C.3. If $\mathbf{D_r[p,s]}$ 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. A total number of N_L **D[p,b]** fields shall be present, regardless of the values of the $\mathbf{D_r[p,s]}$ 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
x1	Vertical prediction
0x	Significance coding disabled
1x	Significance coding enabled

NOTE Table C.2 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.4.

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: 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,λ,s] = 1 and I[p,b,λ',s']=1.

Figure C.1 — A valid selection of raw mode override flags

NOTE Figure C.1 demonstrates a valid composition of raw-mode override flags for 5 horizontal and 2 vertical decomposition levels and two precincts. 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. 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 $\mathbf{D_r[p,s]}$ for precinct **p** and packet **s**, length in bytes of the data subpacket **Ldat[p,s]**, length in bytes of the bitplane count subpacket **Lcnt[p,s]**, and length in bytes of the sign subpacket **Lsgn[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.6.		
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 < 32768 && $N_{L,y}$ ≤ 3) {	Depending on the width of the picture, the number of components and the number of vertical decomposition levels, select the syntax of the picture header. See Table A.6 for the definition of W_f, N_c and $N_{L,y}$.		
Ldat[p,s]	Size of the data subpacket in bytes.	u(15)	0–32767
Lcnt[p,s]	Size of the bitplane count subpacket in bytes	u(13)	0–8191
Lsgn[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.6.	u(11)	0–2047
} else {	End of short packet header		
Ldat[p,s]	Size of the data subpacket in bytes.	u(20)	0–1048575
Lcnt[p,s]	Size of the bitplane count subpacket in bytes	u(20)	0–1048575
Lsgn[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.6.	u(15)	0–32767
}	End of condition for packet header size		
}	End of test for non-empty packet		
}			

C.4 Packet body

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,λ,b,g], the quantization index magnitudes v[**p,λ,b,x**] or

quantization index signs **s[p,λ,b,x]** of a single line **λ** and one or multiple bands **b** of precinct **p**. Table C.4 specifies the syntax of the packet body for precinct **p** and packet **s**.

Input: precinct index **p** and packet index **s**.

Output: bitplane counts **M[p,λ,b,g],** quantization index magnitudes **v[p,λ,b,x]** and quantization index signs **s[p,λ,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 Lcnt[p,s] bytes.	
unpack_data(p,s);	Decode wavelet magnitude data. This subpacket includes Ldat[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.6 for the definition of Fs.	
unpack_signs(p,s);	Decode wavelet magnitude data. This subpacket includes Lsgn[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 precinct 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 precinct header. Depending on configuration, not all subpackets may be present.

C.5.2 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. Table C.5 specifies the syntax of the significance subpacket.

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,λ,b,j] of all significance groups and all bands of the given precinct **p** and packet **s**.

Table C.5 — Syntax of the significance inclusion information

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.6.		
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.8.		
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

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.

Furthermore, the codestream shall be constructed in such a way that for all bands **b**, the sum of the sizes of all bitplane count 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_{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 be the sum of sizes of all bitplane count subpackets contributing to band **b** in precinct **p**:

$$L'_{cnt}[p,b] := \sum_{s=0}^{N_{pc}-1} \sum_{\lambda=L_0[p,b]}^{L_1[p,b]-1} I[p,b,\lambda,s] \times L_{cnt}[p,s]$$

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

$$L_{cnt}^{raw}[p,b] := \sum_{s=0}^{N_{pc}-1} \sum_{\lambda=L_0[p,b]}^{L_1[p,b]-1} I[p,b,\lambda,s] \times \sum_{b'=0}^{N_l-1} \frac{I[p,b',\lambda,s] \times N_{cg}[p,b'] \times B_r}{8}$$

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

NOTE 1 This condition ensures an upper bound for the buffer size a decoder has to reserve for entropy-coded bitplane count data. An encoder can always satisfy this condition by selecting the raw mode for the bitplane count coding mode if the size of the bitplane count subpacket becomes too large.

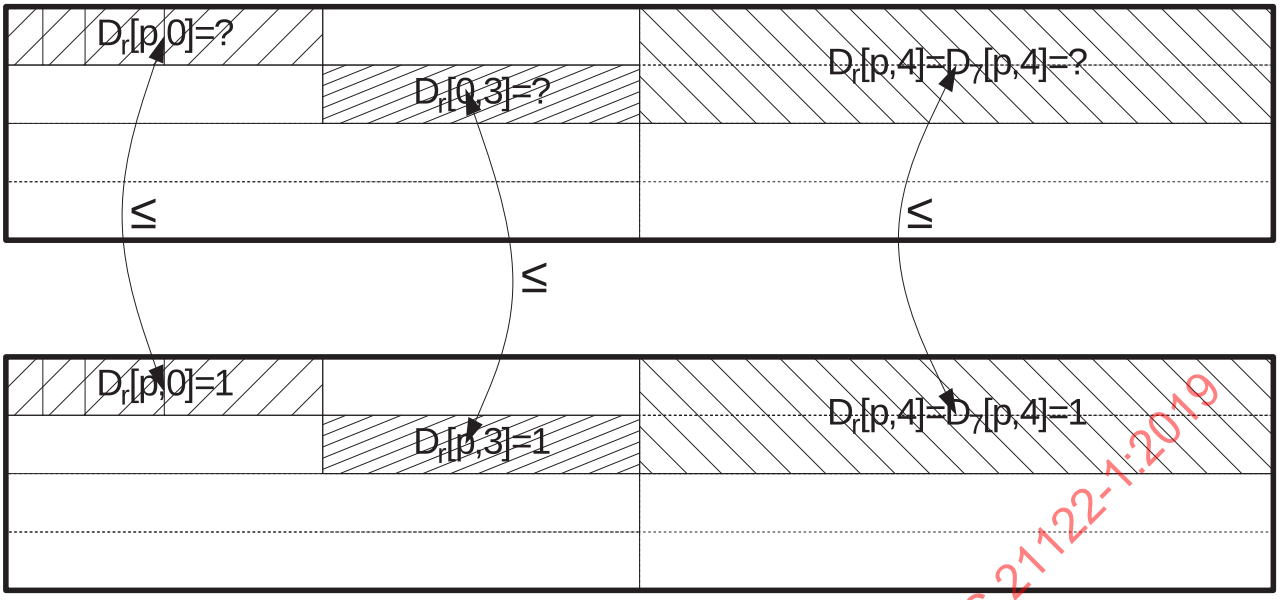


Figure C.2 — Rate constraints for band and packet mode selections

NOTE 2 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.

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
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[b,p];λ<L_1[b,p];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.6.
bytesize = bytesize + Lcnt[p,b]	Number of bytes required for this subpacket
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

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.7.
}	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
}	

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 3 The number of filler bytes at the end of the bitplane count subpacket can be inferred from the Lcnt[p,s] field of the packet header.

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

Output: bitplane counts **M[p,λ,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.6.		
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		

Table C.7 *(continued)*

Name	Notes	Size	Reference
padding	Pad to the next byte boundary	pad(8)	
filler bytes	Arbitrary number of filler bytes	fill()	
}			

C.5.4 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. Table C.8 specifies the syntax of the data subpacket.

NOTE The number of filler bytes at the end of the data subpacket can be inferred from the Ldat[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.6.		
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 sign packing is enabled.		
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.7.		
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
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		

Table C.8 *(continued)*

Name	Notes	Size	Values
}	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

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. Table C.9 specifies the syntax of the sign subpacket.

NOTE The number of filler bytes at the end of the sign subpacket can be inferred from the Lsgn[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,λ,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.6.		
for(g=0;g<N_{cg}[p,b];g=g+1) {	Include data for all groups in this precinct and line. See B.7 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		
}	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()	
}			

C.6 Bitplane count decoding

C.6.1 Bitplane count decoding general provisions

The bitplane count decoding process decodes the bitplane counts **M[p,λ,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 $\mathbf{D_r[p,s]}$ 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 $\mathbf{M_{top}[p,\lambda,b,g]}$ and truncation position $\mathbf{T_{top}[p,b]}$ of the line directly above the current line within the same band. The computation of $\mathbf{M_{top}[p,\lambda,b,g]}$ and $\mathbf{T_{top}[p,b]}$ 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.4.7. 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 $\mathbf{M_{top}[p,\lambda,b,g]}$ and the vertical truncation predictor $\mathbf{T_{top}[p,\lambda,b,g]}$. For that, it requires the first line $\mathbf{L_0[b,p]}$ and the last line $\mathbf{L_1[b,p]}$ of band **b** and precinct **p,** where $\mathbf{L_0[b,p]}$ and $\mathbf{L_1[b,p]}$ are specified in subclause B.5.

Input: precinct index **p**, line index **λ**, band index **b**, bitplane counts **M[p,λ,b,g]** of precinct **p** and precinct $\mathbf{p-N_{p,x}}$, first line $\mathbf{L_0[b,p]}$ and last line $\mathbf{L_1[b,p]}$ of band **b** of precinct **p**.

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

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–1 < L_0[b,p]) {	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[b,p]–1,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–1,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–1 < L_0[b,p]) {	Check whether the given line is the first line of the band in the precinct
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 $\mathbf{D_r[p,s]}$ in the packet header is 1, indicating the raw mode. The bitplane count subpacket contains in this case bitplane counts **M[p,λ,b,g]** directly, using $\mathbf{B_r}$ 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,λ,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.7.		
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 $\mathbf{D_r[p,s]}$ 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,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.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.7.		
t = max(T[p,b],T_{top}[p,b])	Compute effective truncation position for prediction		
m_{top} = max(M_{top}[p,λ,b,g],t)	Compute predictor		
if ((D[p,b] & 2) == 0 || Z[p,λ,b,$\lfloor g/S_s \rfloor$] == 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])	Prediction residual encoded with variable length code	vlc(r,t)	
} 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 $\mathbf{D_r[p,s]}$ 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.7.		
m_{top} = T[p,b]	Set predictor to the truncation position		
if ((D[p,b] & 2) == 0 || Z[p,λ,b,$\lfloor g/S_s \rfloor$] == 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])	prediction residual encoded with variable length code	vlc(r,t)	
} else {			
Δm=0	Non-significant groups have bitplane count T[p,b].		
}	End of significance included		
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 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 outside 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	Increment x		
}			
} 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 > 2θ) {	Check whether this is the unary sub-alphabet		
return x−θ			
} else if (x > 0) {	Check for non-zero symbol, signed sub-alphabet		
if (x & 1) {	Check for an odd codeword		
return -⌈x/2⌉	Return a negative value for odd codewords		
} else {			
return ⌊x/2⌋	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 whether we are in the unary sub-alphabet
n = x + θ	Compute the number of one-bits to write in the unary case
} else {	
x = x × 2	Two bits per symbol in the binary sub-alphabet

Table C.16 *(continued)*

Syntax	Semantics
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
}	
}	
for(i=0;i<n;i=i+1) {	Write out a sequence of n one-bits
out(1)	Write out a single 1-bit
}	
out(0)	Write out 0 as the comma bit
}	
}	

Annex D
(normative)

Quantization

D.1 General

Inverse quantization computes the wavelet coefficients **c[p,λ,b,x]** in all precincts **p**, lines **λ**, bands **p** and positions **x** from the decoded quantization index magnitudes **v[p,λ,b,x]** and their signs **s[p,λ,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,λ,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,λ,b,g]** of the precinct, line and band and quantization index magnitudes **v[p,λ,b,x]** and their signs **s[p,λ,b,x]**.

Output: wavelet coefficients **c[p,λ,b,x]**

Table D.1 — Inverse deadzone quantization

Syntax	Notes
deadzone_dequant(p,λ,b) {	
for(x=0;g<W_{pb}[p,b];x=x+1) {	Iterate over all coefficients of band b in precinct p
g = ⌊x/Ng⌋	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 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,λ,b,g]** of the precinct, line and band and quantization index magnitudes **v[p,λ,b,x]** and their signs **s[p,λ,b,x]**.

Output: wavelet coefficients **c[p,λ,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 σ×v[p,λ,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, 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;g<W_{pb}[p,b];x=x+1) {	Iterate over all coefficients of band b in precinct p
g = ⌊x/Ng⌋	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,λ,b,g]** of the precinct, line and band and wavelet coefficients **c[p,λ,b,x]**.

Output: quantization index magnitudes **v[p,λ,b,x]** and their signs **s[p,λ,b,x]**

Table D.3 — Deadzone quantization

Syntax	Notes
deadzone_quant(p,λ,b) {	
for(x=0;g<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,λ,b,g]** of the precinct, line and band and wavelet coefficients **c[p,λ,b,x]**.

Output: quantization index magnitudes **v[p,λ,b,x]** and and their signs **s[p,λ,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;g<W_{pb}[p,b];x=x+1) {	Iterate over all coefficients of band b in precinct p
g = $\lfloor x/Ng \rfloor$	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
} else {	Coefficient does not include sufficient number of bitplanes
s[p,b,λ,x] = 0	
v[p,b,λ,x] = 0	Quantize to zero

Table D.4 *(continued)*

Syntax	Notes
}	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,λ,b,g]** from the wavelet coefficients **c[p,λ,b,g]**. The bitplane counts **M[p,λ,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,λ,b,x]**.

Output: bitplane counts **M[p,λ,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,λ,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,λ,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($N_{L,x}$,$N_{L,y}$)	Compute number of initial horizontal transformations to perform
for(d_x=$N_{L,x}$;d_x>D_x;d_x=d_x−1) {	Loop over horizontal decomposition levels
hor_transform(k,$LL_{dx-1,NLy}$, $LL_{dx,NLy}$,$HL_{dx,NLy}$)	Horizontally transform the $LL_{dx,NLy}$ and $HL_{dx,NLy}$ bands of component k into the $LL_{dx-1,NLy}$ 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
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

Table E.1 *(continued)*

Syntax	Notes
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,λ,b,x] from all precincts and component **k** to temporary bands T[β,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** and N_β are defined in subclause B.3

Input: component index **k** and inversely quantized wavelet coefficients c[p,λ,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[β,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
b= k+N_c×β	Compute the band index b from the filter type β and the component index k
for(y=0;y<H_b[b];y=y+1) {	Iterate over all rows of band b
for(x=0;x<W_b[b];x=x+1) {	Iterate over all colums of band b
$p = N_{p,x} \times \left\lfloor \frac{y \times s_y[k] \times 2^{d_y[\beta]}}{2^{N_{L,y}}} \right\rfloor + \left\lfloor \frac{x \times s_x[k] \times 2^{d_x[\beta]}}{C_s} \right\rfloor$	Compute the precinct index p from the horizontal position x and vertical position y.
$\lambda = y \bmod 2^{N_{L,y} - d_y[\beta]}$	Compute the line within the precinct from the vertical position y
$\xi = x \bmod \left\lceil \frac{C_s}{s_x[k] \times 2^{d_x[\beta]}} \right\rceil$	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 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[β_L,x,y] and T[β_H,x,y].

Output: wavelet coefficients in temporary output band T[β_0,x,y]

Table E.3 — Horizontal inverse wavelet transformation

Syntax	Notes
hor_transform(k,β_o,β_L,β_H) {	Horizontally inverse transform the low-pass coefficients β_L and high-pass coefficients β_H to the output band β_o.
b= k+N_c×β_o	Compute the band index b from the filter type β and the component index k
for(y=0;y<H_b[b];y=y+1) {	Iterate over all rows of band b
for(x=0;x<W_b[b];x=x+1) {	Iterate over all columns of band b
i = $\lfloor x/2 \rfloor$	Compute the input sample position in the source band
if (x umod 2 = 0) {	if even sample position
X[x]=T[β_L,i,y]	Assign the low-pass input to the even samples of the temporary array X
} else {	else odd sample position
X[x]=T[β_H,i,y]	Assign the high-pass samples to the odd samples of the temporary array X
}	End of check for even/odd coefficients
}	End of loop over columns
extend_samples(W_b[b])	Symmetrically extend the samples X across the boundary
inverse_filter_1D(W_b[b])	Performs an inverse filtering on the temporary array X
for(x=0;x<W_b[b];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[β_L,x,y] and T[β_H,x,y].

Output: wavelet coefficients in temporary output band T[β_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.
b= k+N_β×β_o	Compute the band index b from the filter type β and the component index k
for(x=0;x<W_b[b];x=x+1) {	Iterate over all columns of band b
for(y=0;y<H_b[b];y=y+1) {	Iterate over all rows of band b
i = $\lfloor y/2 \rfloor$	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

Table E.4 *(continued)*

Syntax	Notes
X[y]=T[β_H,y,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[b])	Symmetrically extend the samples X across the boundary
inverse_filter_1D(H_b[b]);	Performs an inverse filtering on the temporary array X
for(y=0;y<H_b[b];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

Table E.6 *(continued)*

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

E.8 Assignment of output coefficients

This step assigns 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[β,x,y].

Table E.7 — Output assignment

Syntax	Notes
assign_output(k,β) {	Assign the output of component c 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[β,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,λ,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,λ,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($N_{L,x}$,$N_{L,y}$)	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