
**Information technology — JPEG XL
image coding system —**

**Part 1:
Core coding system**

*Technologies de l'information — Système de codage d'images
JPEG XL —*

Partie 1: Système de codage de noyau



Reference number
ISO/IEC 18181-1:2022(E)

Documents
AVA Document Sharing Hub
© ISO/IEC 2022



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2022

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

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

Published in Switzerland

Contents

	Page
Foreword.....	v
Introduction.....	vi
1 Scope.....	1
2 Normative references.....	1
3 Terms and definitions.....	1
3.1 Data storage.....	2
3.2 Inputs.....	2
3.3 Processes.....	3
3.4 Image organization.....	4
3.5 DCT.....	5
4 Abbreviated terms.....	6
5 Conventions.....	6
5.1 Mathematical symbols.....	6
5.2 Functions.....	6
5.3 Operators.....	7
5.4 Pseudocode.....	7
6 Functional concepts.....	8
6.1 Image organization.....	8
6.2 Group splitting.....	8
6.3 Codestream and bitstream.....	9
6.4 Multiple frames.....	10
6.5 Mirroring.....	10
7 Encoder requirements.....	10
8 Decoder requirements.....	10
9 Codestream.....	10
9.1 Syntax.....	10
9.1.1 Reading a field.....	11
9.1.2 Initializing a field.....	11
9.2 Field types.....	11
9.2.1 u(n).....	11
9.2.2 U32(d0, d1, d2, d3).....	11
9.2.3 U64().....	11
9.2.4 Varint().....	12
9.2.5 U8().....	12
9.2.6 F16().....	12
9.2.7 Bool().....	12
9.2.8 Enum(EnumTable).....	12
9.2.9 ZeroPadToByte().....	13
9.3 Structure.....	13
10 Decoding process.....	13
Annex A (normative) Headers.....	15
Annex B (normative) ICC profile.....	25
Annex C (normative) Frames.....	32
Annex D (normative) Entropy decoding.....	58
Annex E (normative) Weighted predictor.....	67
Annex F (normative) Adaptive quantization.....	70
Annex G (normative) Chroma from luma.....	71

Annex H (normative) Extensions	72
Annex I (normative) Integral transforms	73
Annex J (normative) Restoration filters	84
Annex K (normative) Image features	87
Annex L (normative) Colour transforms	92
Annex M (informative) Encoder overview	98
Bibliography	101

Foreword

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

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

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

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

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

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

A list of all parts in the ISO/IEC 18181 series can be found on the ISO and IEC websites.

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

Introduction

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of 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 they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

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

Information technology — JPEG XL image coding system —

Part 1: Core coding system

1 Scope

This document defines a set of compression methods for coding one or more images of bi-level, continuous-tone greyscale, or continuous-tone colour, or multichannel digital samples.

This document:

- specifies decoding processes for converting compressed image data to reconstructed image data;
- specifies a codestream syntax containing information for interpreting the compressed image data;
- provides guidance on encoding processes for converting source image data to compressed image data.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 15076-1:2010, *Image technology colour management — Architecture, profile format and data structure — Part 1: Based on ICC.1:2010*

ISO/IEC 60559, *Information technology — Microprocessor Systems — Floating-Point arithmetic*

IEC 61966-2-1, *Multimedia systems and equipment — Colour measurement and management — Part 2-1: Colour management — Default RGB colour space — sRGB*

Rec. ITU-R BT.2100-2, *Image parameter values for high dynamic range television for use in production and international programme exchange*

Rec. ITU-R BT.709-6, *Parameter values for the HDTV standards for production and international programme exchange*

SMPTE ST 428-1, *D-Cinema distribution master — Image characteristics*

3 Terms and definitions

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

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

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

3.1 Data storage

3.1.1

byte

8 consecutive bits encoding a value between 0 and 255

3.1.2

big endian

value representation with bytes in most to least-significant order

3.1.3

bitstream

sequence of bytes from which bits are read starting from the least-significant bit of the first byte

3.1.4

codestream

bitstream representing compressed image data

3.1.5

bundle

structured data consisting of one or more fields

3.1.6

field

numerical value or bundle, or an array of either

3.1.7

histogram

array of unsigned integers representing a probability distribution, used for entropy coding

3.1.8

set

unordered collection of elements

3.2 Inputs

3.2.1

pixel

vector of dimension corresponding to the number of channels, consisting of samples

3.2.2

sample

integer or real value, of which there is one per channel per pixel

3.2.3

greyscale

image representation in which each pixel is defined by a single sample representing intensity (either luminance or luma depending on the ICC profile)

3.2.4

continuous-tone image

image having samples consisting of more than one bit

3.2.5

opsin

photosensitive pigments in the human retina, having dynamics approximated by the XYZ colour space

3.2.6

burst

sequences of images typically captured with identical settings

3.2.7**animation**

series of pictures and timing delays to display as a video medium

3.2.8**composite**

series of images that are superimposed

3.2.9**frame**

single image (possibly part of a burst or animation or composite)

3.2.10**preview**

lower-fidelity rendition of one of the frames (e.g. lower resolution), or a frame that represents the entire content of all frames

3.3 Processes**3.3.1****decoding process**

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

3.3.2**decoder**

embodiment of a decoding process

3.3.3**encoding process**

process which takes as its input continuous-tone image(s) and outputs compressed image data in the form of a codestream

3.3.4**encoder**

embodiment of an encoding process

3.3.5**lossless**

descriptive term for encoding and decoding processes in which the output of a decoding procedure is identical to the input to the encoding procedure

3.3.6**lossy**

descriptive term for encoding and decoding processes which are not lossless

3.3.7**upsampling**

procedure by which the (nominal) spatial resolution of a channel is increased

3.3.8**downsampling**

procedure by which the spatial resolution of a channel is reduced

3.3.9**entropy encoding**

lossless procedure designed to convert a sequence of input symbols into a sequence of bits such that the average number of bits per symbol approaches the entropy of the input symbols

3.3.10**entropy encoder**

embodiment of an entropy encoding procedure

3.3.11

entropy decoding

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

3.3.12

entropy decoder

embodiment of an entropy decoding procedure

3.3.13

Gabor-like transform

convolution with default or signalled 3x3 kernel for deblocking

3.3.14

tick

unit of time such that animation frame durations are integer multiples of the tick duration

3.4 Image organization

3.4.1

grid

2-dimensional array; $a[x, y]$ means addressing an element of grid a at row y and column x . Where so specified, addressing elements with coordinates outside of bounding rectangle ($x < 0$, or $y < 0$, or $x \geq \text{width}$, or $y \geq \text{height}$) is allowed

3.4.2

sample grid

common coordinate system for all samples of an image, with top-left coordinates (0, 0), the first coordinate increasing towards the right, and the second increasing towards the bottom

3.4.3

channel

component

rectangular array of samples having the same designation, regularly aligned along a sample grid

3.4.4

rectangle

rectangular area within a channel or grid

3.4.5

width

width in samples of a sample grid or a rectangle

3.4.6

height

height in samples of a sample grid or a rectangle

3.4.7

raster order

access pattern from left to right in the top row, then in the row below and so on

3.4.8

naturally aligned

positioning of a power-of-two sized rectangle such that its top and left coordinates are divisible by its width and height, respectively

3.4.9

block

naturally aligned square rectangle covering up to 8×8 input pixels

3.4.10**group**

naturally aligned square rectangle covering up to $2^n \times 2^n$ (with n between 7 and 10, inclusive) input pixels

3.4.11**table of contents**

data structure that enables seeking to a group or the next frame within a codestream

3.4.12**section**

part of the codestream with an offset and length that are stored in a frame's table of contents

3.5 DCT**3.5.1****coefficient**

input value to the inverse DCT

3.5.2**quantization**

method of reducing the precision of individual coefficients

3.5.3**varblock**

variable-size rectangle of input pixels

3.5.4**dct_block**

an array with 64 elements corresponding to DCT coefficients of a (8×8) block

3.5.5**var-DCT**

lossy encoding of a frame that applies DCT to varblocks

3.5.6**LF coefficient**

lowest frequency DCT coefficient, containing the average value of a block or the lowest-frequency coefficient within the 8×8 rectangle of a varblock of size greater than 8×8

3.5.7**HF coefficients**

all DCT coefficients apart from the LF coefficients, i.e. the high frequency coefficients

3.5.8**pass**

data enabling decoding of successively higher resolutions

3.5.9**LF group**

$2^n \times 2^n$ LF values from a naturally aligned rectangle covering up to $2^{n+3} \times 2^{n+3}$ input pixels

3.5.10**quantization weight**

factor that a quantized coefficient is multiplied by prior to application of the inverse DCT in the decoding process

3.5.11**channel decorrelation**

method of reducing total encoded entropy by removing correlations between channels

3.5.12

channel correlation factor

factor by which a channel should be multiplied by before adding it to another channel to undo the channel decorrelation process

4 Abbreviated terms

DCT: discrete cosine transform (DCT-II as specified in [L.2](#))

IDCT: inverse discrete cosine transform (DCT-III as specified in [L.2](#))

LF: $N / 8 \times M / 8$ square of lowest frequency coefficients of $N \times M$ DCT coefficients

RGB: additive colour model with red, green, blue channels

LMS: absolute colour space representing the response of cone cells in the human eye

XYB: absolute colour space based on gamma-corrected LMS, in which X is derived from the difference between L and M, Y is an average of L and M (behaves similarly to luminance), and B is derived from the S ("blue") channel

5 Conventions

5.1 Mathematical symbols

$[a, b], (c, d), [e, f)$	closed or open or half-open intervals containing all integers or real numbers x (depending on context) such that $a \leq x \leq b, c < x < d, e \leq x < f.$
$\{a, b, c\}$	ordered sequence of elements
π	the smallest positive zero of the sine function

5.2 Functions

$\text{sqrt}(x)$	square root, such that $(\text{sqrt}(x))^2 == x$ and $\text{sqrt}(x) \geq 0$. Undefined for $x < 0$.
$\text{cbrt}(x)$	cube root, such that $(\text{cbrt}(x))^3 == x$.
$\cos(r)$	cosine of the angle r (in radians)
$\text{erf}(x)$	Gauss error function: $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$
$\log(x)$	natural logarithm of x . Undefined for $x \leq 0$.
$\log_2(x)$	base-two logarithm of x . Undefined for $x \leq 0$.
$\text{floor}(x)$	the largest integer that is less than or equal to x
$\text{ceil}(x)$	the smallest integer that is greater than or equal to x
$\text{abs}(x)$	absolute value of x : equal to $-x$ if $x < 0$, otherwise x
$\text{sign}(x)$	sign of x , 0 if x is 0, +1 if x is positive, -1 if x is negative
$\text{UnpackSigned}(u)$	equivalent to $u / 2$ if u is even, and $-(u + 1) / 2$ if u is odd

<code>clamp(x, lo, hi)</code>	equivalent to <code>min({max({lo, x}), hi})</code>
<code>InterpretAsF16(u)</code>	the real number resulting from interpreting the unsigned 16-bit integer <code>u</code> as a binary16 floating-point number representation (cf. ISO/IEC 60559)
<code>InterpretAsF32(u)</code>	the real number resulting from interpreting the unsigned 32-bit integer <code>u</code> as a binary32 floating-point number representation (cf. ISO/IEC 60559)
<code>len(a)</code>	length (number of elements) of array <code>a</code>
<code>sum(a)</code>	sum of all elements of the array/tuple/sequence <code>a</code>
<code>max(a)</code>	maximal element of the array/tuple/sequence <code>a</code>
<code>min(a)</code>	smallest element of the array/tuple/sequence <code>a</code>

5.3 Operators

This document uses the operators defined by the C++ programming language [2], with the following differences:

<code>*</code>	multiplication
<code>*=</code>	<code>a *= b</code> is equivalent to <code>a = a * b</code>
<code>/</code>	division of real numbers without truncation or rounding. Division by zero is undefined.
<code>x^y</code>	exponentiation, <code>x</code> to the power of <code>y</code>
<code><<</code>	left shift: <code>x << s</code> is defined as <code>x × 2^s</code>
<code>>></code>	right shift: <code>x >> s</code> is defined as <code>floor(x / 2^s)</code>
<code>Umod</code>	<code>a Umod d</code> is the unique integer <code>r</code> in <code>[0, d)</code> for which <code>a == r + q × d</code> for a suitable integer <code>q</code>
<code>Idiv</code>	<code>a Idiv b</code> is equivalent to <code>a / b</code> , rounded towards zero to an integer value

The order of precedence for these operators is listed below in descending order. 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
<code>++x, --x</code>	prefix increment/decrement	right to left
<code>x^y</code>	exponentiation	right to left
<code>!, ~</code>	logical/bitwise NOT	right to left
<code>*, /, Idiv, Umod</code>	multiplication, division, integer division, remainder	left to right
<code>+, -</code>	addition and subtraction	left to right
<code><<, >></code>	left shift and right shift	left to right
<code><, >, <=, >=</code>	relational	left to right
<code>=</code>	assignment	right to left
<code>+=, -=, *=</code>	compound assignment	right to left

5.4 Pseudocode

This document describes functionality using pseudocode formatted as follows:

```
// Informative comment
var = u(8); // Defined in 9.2.1
if (var == 1) return; // Stop executing this code snippet
[[Normative specification: var != 0]]
(out1, out2) = Function(var, kConstant);
```

Variables such as `var` are typically referenced by text outside the source code.

The semantics of this pseudocode are those of the C++ programming language [2], with the following exceptions:

- Symbols from 5.1 and functions from 5.2 are allowed;
- Multiplication, division, remainder and exponentiation are expressed as specified in 5.3;
- Functions can return tuples which unpack to variables as in the above example;
- `[[ ]]` enclose normative directives specified using prose;
- All integers are stored using two's complement;
- Expressions and variables of which types are omitted, are understood as real numbers.

Where unsigned integer wraparound and truncated division are required, `Umod` and `Idiv` (see 5.3) are used for those purposes.

Numbers with a `0x` prefix are in base 16 (hexadecimal), and apostrophe (`'`) characters inside them are understood to have no effect.

EXAMPLE `0x0001'0000 == 65536.`

6 Functional concepts

6.1 Image organization

A channel is defined as a rectangular array of (integer or real) samples regularly aligned along a sample grid of `width` sample positions horizontally and `height` sample positions vertically. The number of channels may be 1 to 4099 (see `num_extra_channels` in A.6).

A pixel is defined as a vector of dimension corresponding to the number of channels, consisting of samples with a position matching that of the pixel. The index of a sample is numbered from 0 to number of channels - 1.

An image is defined as the two-dimensional array of pixels, and its width is `width` and height is `height`. Unless otherwise mentioned, channels are accessed in the following "raster order": left to right column within the topmost row, then left to right column within the row below the top, and so on until the rightmost column of the bottom row.

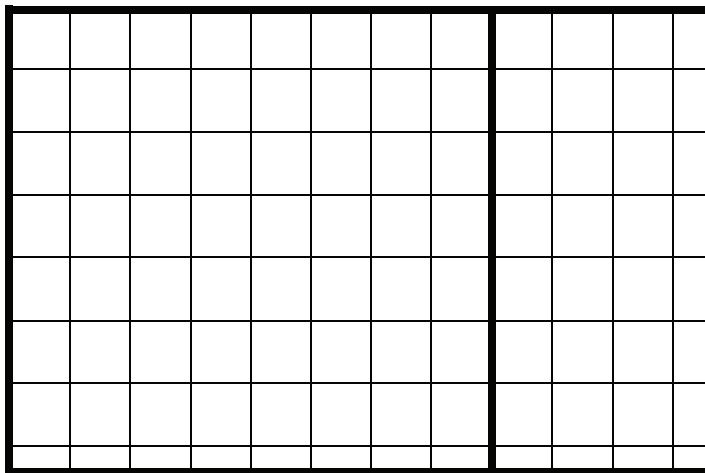
6.2 Group splitting

Channels are logically partitioned into naturally-aligned groups of `kGroupDim` × `kGroupDim` samples. The effective dimension of a group (i.e. how many pixels to read) can be smaller than `kGroupDim` for groups on the right or bottom of the image. The decoder ensures the decoded image has the dimensions specified in `SizeHeader` by cropping at the right and bottom as necessary. Unless otherwise specified, `kGroupDim` is 256.

LF groups likewise consist of `kGroupDim` × `kGroupDim` LF samples, with the possibility of a smaller effective size on the right and bottom of the image.

Groups can be decoded independently. A 'table of contents' stores the size (in bytes) of each group to allow seeking to any group. An optional permutation allows groups to be arranged in arbitrary order within the codestream.

EXAMPLE [Figure 1](#) shows an example of the HF groups and LF groups of an image.



Frame: 2970×1868 pixels

HF groups:

11×7 groups of 256×256 pixels,

1×7 groups of 154×256 pixels,

7×1 groups of 256×76 pixels,

1 group of 154×76 pixels

LF groups:

1 group of 256×233 LF coefficients

(covering 2048×1868 pixels),

1 group of 116×233 LF coefficients

(covering 922×1868 pixels)

Figure 1 — Group splitting example

6.3 Codestream and bitstream

A bitstream is a finite sequence of bytes. A codestream is a bitstream that represent compressed image data and metadata. N bytes can also be viewed as $8 \times N$ bits. The first 8 bits are the bits constituting the first byte, in least to most significant order, the next eight bits (again in least to most significant order) constitute the second byte, and so on. Unless otherwise specified, bits are read from the codestream as specified in [9.2.1](#).

NOTE Ordering bits from least to most significant allows using special CPU instructions to isolate the least-significant bits.

Subsequent Annexes or subclauses indicate some elements of the codestream are byte-aligned. For such elements, the decoder takes actions before and after reading the element as follows. Immediately before encountering the element, the decoder invokes ZeroPadToByte() ([9.2.9](#)). After finishing reading the element, the decoder invokes ZeroPadToByte() ([9.2.9](#)).

ZeroPadToByte specifies that the padding bits, if any, are zero for the codestream to be valid. This can serve as an additional indicator of codestream integrity.

6.4 Multiple frames

A codestream may contain multiple frames. These can constitute an animation, a burst (arbitrary images with identical dimensions), or a composite still image with one or more frames rendered on top of the first frame.

NOTE The frame that is being decoded is referred to as the current frame.

6.5 Mirroring

Some operations access samples with coordinates c_x , c_y that are outside the image bounds. The decoder redirects such accesses to a valid sample at the coordinates $\text{Mirror1D}(c_x, \text{width})$, $\text{Mirror1D}(c_y, \text{height})$, defined in the following code:

```
Mirror1D(coord, size) {
    if (coord < 0) return Mirror1D(-coord - 1, size);
    else if (coord >= size) return Mirror1D(2 * size - 1 - coord, size);
    else return coord;
}
```

7 Encoder requirements

An encoder is an embodiment of the encoding process. This document does not specify an encoding process, and any encoding process is acceptable as long as the codestream conforms to the codestream syntax specified in this document. [Annex M](#) provides an informative description of such an encoding process.

8 Decoder requirements

A decoder is an embodiment of the decoding process. The decoder reconstructs sample values (arranged on a rectangular sampling grid) from a codestream as specified in this document. [Annexes A](#) to [L](#) are normative in the sense that they are defining an output that alternative implementations shall duplicate.

9 Codestream

9.1 Syntax

The codestream is organized into "bundles" consisting of one or more conceptually related "fields". A field can be a bundle, or an array/value of a type from [9.2](#). The graph of contained-within relationships between types of bundles is acyclic. This document specifies the structure of each bundle using tables structured as in [Table 1](#), with one row per field (in top to bottom order).

Table 1 — Structure of a table describing a bundle

condition	type	default	name
-----------	------	---------	------

If `condition` is blank or evaluates to true, the field is read from the codestream ([9.1.1](#)). Otherwise, the field is instead initialized to `default` ([9.1.2](#)).

A `condition` of the form `for(i = 0; condition; ++i)` is equivalent to replacing this row with a sequence of rows obtained from the current one by removing its condition and replacing the value of `i`

in each column with the consecutive values assumed by the loop-variable `i` until `condition` is false. If `condition` is initially false, the current row has no effect.

If `name` ends with `[n]` then the field is a fixed-length array with `n` entries. If `condition` is blank or evaluates to true, each array element is read from the codestream (9.1.1) in order of increasing index. Otherwise, each element is initialized to `default` (9.1.2). The `name` is potentially referenced in the `condition` or `type` of a subsequent row, or the `condition` of the same row.

9.1.1 Reading a field

If a field is to be read from the codestream, the `type` entry determines how to do so. If it is a basic field type, it is read as described in 9.2. Otherwise `type` is a (nested) bundle denoted `Nested`, residing within a parent bundle `Parent`. `Nested` is read as if the rows of the table defining `Nested` were inserted into the table defining `Parent` in place of the row that defined `Nested`. This principle is applied recursively, corresponding to a depth-first traversal of all fields.

9.1.2 Initializing a field

If a field is to be initialized to `default`, the `type` entry determines how to do so. If it is a bundle, then `default` is blank and each field of the bundle is (recursively) initialized to the `default` specified within the bundle's table. Otherwise, if `default` is not blank, the field is set to `default`, which is a valid value of the same `type`.

9.2 Field types

9.2.1 `u(n)`

`u(0)` evaluates to the value zero without reading any bits. For $n > 0$, `u(n)` reads n bits from the codestream, advances the position accordingly, and returns the value in the range $[0, 2^n)$ represented by the bits. The decoder first reads the least-significant bit of the value, from the least-significant not yet consumed bit in the first not yet fully consumed byte of the codestream. The next bit of the value (in increasing order of significance) is read from the next (in increasing order of significance) bit of the same byte unless all its bits have already been consumed, in which case the decoder reads from the least-significant bit of the next byte, and so on.

9.2.2 `U32(d0, d1, d2, d3)`

In this subclause, 'distribution' refers to one of the following three encodings of a range of values: `Val(u)`, `Bits(n)`, or `BitsOffset(n, offset)`. The `d0`, `d1`, `d2`, `d3` parameters represent distributions.

`U32(d0, d1, d2, d3)` reads an unsigned 32-bit `value` in $[0, 2^{32})$ as follows. The decoder first reads a `u(2)` from the codestream indicating which distribution to use (0 selects `d0`, 2 selects `d2` etc.). Let `d` denote this distribution, which determines how to decode `value`.

If `d` is `Val(u)`, `value` is the integer `u`. If `d` is `Bits(n)`, `value` is read as `u(n)`. If `d` is `BitsOffset(n, offset)`, the decoder reads $v = u(n)$. The resulting `value` is $(offset + v) \text{ Umod } 2^{32}$.

NOTE The value of `u` is implicitly defined by `Val(u)` and not stored in the codestream.

EXAMPLE For a field of type `U32(Val(8), Val(16), Val(32), Bits(7))`, the bits 10 result in `value = 32`. For a `U32(Bits(2), Bits(4), Bits(6), Bits(8))` field, the bits 010111 result in `value = 7`.

9.2.3 `U64()`

`U64()` reads an unsigned 64-bit `value` in $[0, 2^{64})$ using a single variable-length encoding. The decoder first reads a `u(2)` selector `s`. If `s == 0`, `value` is 0. If `s == 1`, `value` is `BitsOffset(4, 1)` (9.2.2). If `s == 2`, `value` is `BitsOffset(8, 17)`. Otherwise `s == 3` and `value` is read from a 12-bit part, zero or more 8-bit parts, and zero or one 4-bit part as specified by the following code:

```

value = u(12); shift = 12;
while (u(1) == 1) {
    if (shift == 60) {
        value += u(4) << shift; // only 4, we already read 60
        break;
    }
    value += u(8) << shift; shift += 8;
}

```

EXAMPLE the largest possible value ($2^{64} - 1$) is encoded as 73 consecutive 1-bits.

9.2.4 Varint()

Varint() reads an unsigned integer *value* of up to 63 bits as specified by the following code:

```

value = 0; shift = 0;
while (1) {
    b = u(8);
    value += (b & 127) << shift;
    if (b <= 127) break;
    shift += 7;    [[ shift < 63 ]];
}

```

9.2.5 U8()

U8() reads an integer *value* in the range [0, 256) using [1, 12) bits as specified by the following code:

```

if (u(1) == 0) value = 0;
else { n = u(3); value = u(n) + (1 << n); }

```

EXAMPLE The bit 0 results in value 0, bits 1000 result in value 1, bits 10011 in value 3.

9.2.6 F16()

F16() reads a binary16 representation (as specified in ISO/IEC 60559) of a real *value* in [-65504, 65504]. *value* is as specified by the following code:

```

bits16 = u(16);
biased_exp = (bits16 >> 10) & 0x1F;
value = InterpretAsF16(bits16);

```

The value of *biased_exp* is not 31.

NOTE This rules out NaN and infinities.

9.2.7 Bool()

Bool() reads a boolean value as *u(1) ? true : false*.

9.2.8 Enum(EnumTable)

Enum(EnumTable) reads *v* = U32(Val(0), Val(1), BitsOffset(4, 2), BitsOffset(6, 18)). The value *v* does not exceed 63 and is a *value* defined by the table in the subclause titled EnumTable. Such tables are structured according to [Table 2](#), with one row per unique *value*.

Table 2 — Structure of a table describing an enumerated type

name	value	meaning
------	-------	---------

An enumerated type is interpreted as having the meaning of the row where value is v. name (or where ambiguous, EnumTable.name) is an identifier for purposes of referring to the same meaning elsewhere in this document. name begins with a k prefix, e.g. kRGB.

9.2.9 ZeroPadToByte()

The decoder reads a $u(n)$, where n is zero if p , the 0-based index of the next unread bit in the codestream, is a multiple of 8, otherwise $n = 8 - (p \bmod 8)$. The result is equal to zero.

NOTE The effect of ZeroPadToByte() is skip to the next byte boundary if not already at a byte boundary, and require all skipped bits to have value 0.

9.3 Structure

The codestream consists of headers ([Annexes A, H](#)) and an ICC profile, if present ([Annex B](#)), followed by one or more frames ([Annex C](#)), as shown in [Table 3](#). This table and those it references, together with the syntax description ([9.1](#)), specify how a decoder reads the headers and frame(s).

Table 3 — Codestream structure

condition	type	name	Annex
	Headers	headers	A, H
headers.metadata.want_icc		icc	B
headers.metadata.have_preview	Frame	preview_frame	C
	Frame	frames[0]	C
for (i = 1; !frames[i - 1].frame_header.is_last; ++i)	Frame	frames[i]	C

The preview_frame is a self-contained frame whose FrameHeader ([C.2](#)) specifies $lf_level=0$, $frame_type=kRegularFrame$ and $save_as_reference=0$.

10 Decoding process

After reading the header and frame data ([9.3](#) and [Annexes A, B](#) and [C](#)), the decoder can be viewed as a pipeline with multiple stages, as shown in [Figure 2](#).

Some Annexes or subclauses begin with a condition; the decoder only applies the processing steps described in such an Annex or subclause if its condition holds true.

[Annex D](#) specifies how the decoder reads entropy-coded data.

[Annex E](#) specifies how the decoder transforms channels decoded as residuals of a self-correcting weighted predictor into reconstructed samples.

The decoder converts adaptively quantized integers to DCT coefficients as specified in [Annex F](#).

[Annex G](#) specifies how the decoder recorrelates channels stored as differences w.r.t. a linear function of another channel for the purpose of decorrelation ('chroma from luma').

The decoder performs or skips inverse integral transforms as specified in [Annex I](#).

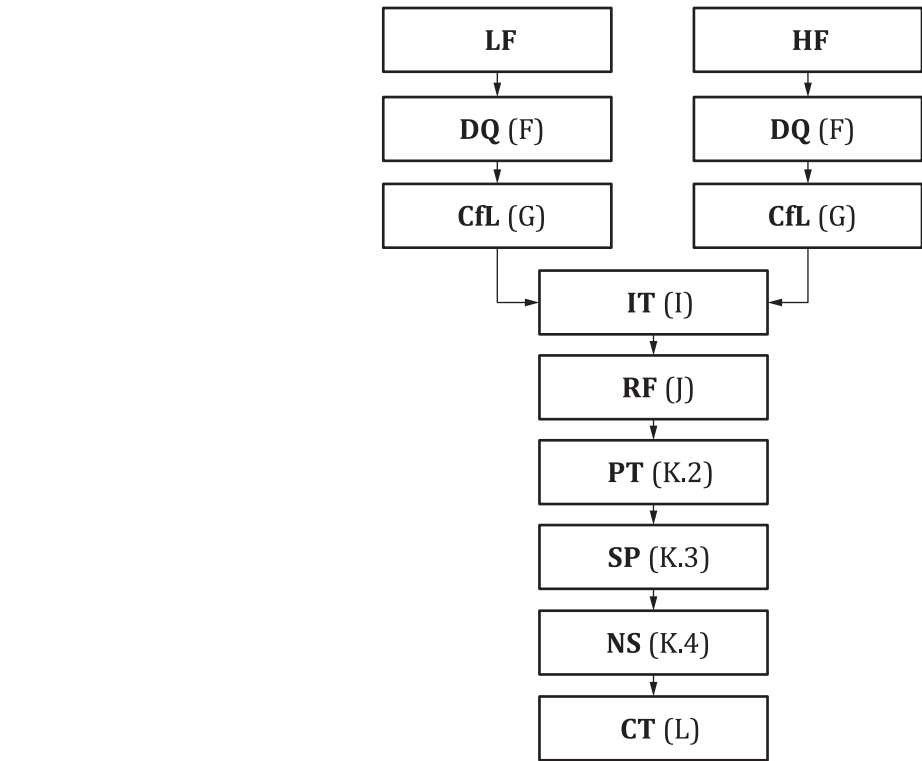
The decoder applies zero, one or two or restoration filters as specified in [Annex J](#).

The presence/absence of additional image features (patches, splines and noise) is indicated in the frame header. The decoder draws these as specified in [Annex K](#). Image features (if present) are rendered after restoration filters (if enabled), in the listed order.

Finally, the decoder performs zero, one or more colour-space transforms as specified in [Annex L](#).

NOTE For an introduction to the coding tools and additional background, refer to the Joint Photographic Experts Group (JPEG) committee JPEG XL website [5].

EXAMPLE The lossless mode does not involve adaptive dequantization.



- Key**
- DQ dequantization
 - Cfl chroma from luma
 - IT integral transform
 - RF restoration filter
 - PT patches
 - SP splines
 - NS noise
 - CT colour transform

Figure 2 — Decoder block diagram after LF/HF coefficients of a frame have been read

Annex A (normative)

Headers

A.1 General

The decoder reads the Headers bundle ([Table A.1](#)) as specified in [9.1](#).

Table A.1 — Headers bundle

condition	type	name	subclause
	Signature	signature	A.2
	SizeHeader	size	A.3
	ImageMetadata	metadata	A.6

A.2 Signature

[Table A.2](#) specifies the Signature bundle.

Table A.2 — Signature bundle

condition	type	default	name
	u(8)		ff
	u(8)		type

ff is 255. type is 10.

A.3 Image dimensions

[Table A.3](#) specifies the SizeHeader bundle.

Table A.3 — SizeHeader bundle

condition	type	default	name
	Bool()	false	small
small	u(5)	0	height_div8_minus_1
!small	U32(Bits(9), Bits(13), Bits(18), Bits(30))	0	height_minus_1
	u(3)	0	ratio
small && ratio == 0	u(5)	0	width_div8_minus_1
!small && ratio == 0	U32(Bits(9), Bits(13), Bits(18), Bits(30))	0	width_minus_1

height is defined as $\text{small} ? (\text{height_div8_minus_1} + 1) \times 8 : \text{height_minus_1} + 1$. width is defined as follows. If $\text{ratio} == 0$, then $\text{width} = \text{small} ? (\text{width_div8_minus_1} + 1) \times 8 : \text{width_minus_1} + 1$. Otherwise, ratio is in the range [1, 8) and width is computed as defined in [Table A.4](#).

NOTE 1 These encodings cannot represent a zero-width or zero-height image.

Table A.4 — AspectRatio

ratio	Meaning
0	width coded separately
1	width = height
2	width = floor(height × 12 / 10)
3	width = floor(height × 4 / 3)
4	width = floor(height × 3 / 2)
5	width = floor(height × 16 / 9)
6	width = floor(height × 5 / 4)
7	width = floor(height × 2 / 1)

If the `ratio` field of `SizeHeader` or `PreviewHeader` != 0, their `width` field are computed from `height` and `ratio` as defined in [Table A.4](#).

NOTE 2 Unlike other bundles, `SizeHeader` provides a guarantee on when the decoder can access its fields. In a streaming use case, it can be helpful for the decoder to know the image dimensions before having received the entire codestream. Once at least 94 bits of the codestream are accessible, or the entire codestream is known to have been transmitted, the decoder can decode `SizeHeader` and access its fields.

[Table A.5](#) specifies the `PreviewHeader` bundle.

Table A.5 — PreviewHeader bundle

condition	type	default	name
	Bool()		div8
div8	U32(Val(16), Val(32), BitsOffset(5, 1), BitsOffset(9, 33))	1	height_div8
!div8	U32(BitsOffset(6, 1), BitsOffset(8, 65), BitsOffset(10, 321), BitsOffset(12, 1345))	8 × height_div8	height
	u(3)		ratio
div8 && ratio == 0	U32(Val(16), Val(32), BitsOffset(5,1), BitsOffset(9, 33))	1	width_div8
!div8 && ratio == 0	U32(BitsOffset(6, 1), BitsOffset(8, 65), BitsOffset(10, 321), BitsOffset(12, 1345))	1	w_explicit

NOTE 3 The presence of `PreviewHeader` is signalled by `metadata.have_preview`.

`width` is defined as follows. If `ratio == 0`, then `width = (div8 ? width_div8 × 8 : w_explicit)`. Otherwise, `ratio` is in the range [1, 8) and `width` is computed as defined in [Table A.4](#). The values of `width` and `height` do not exceed 4096.

NOTE 4 This encoding cannot represent a zero-width or zero-height preview.

A.4 ColourEncoding

The `ColourEncoding` bundle is specified in [Table A.13](#). It references [Table A.6](#) (`Customxy`), [Table A.7](#) (`ColourSpace`), [Table A.8](#) (`WhitePoint`), [Table A.9](#) (`Primaries`), [Table A.10](#) (`TransferFunction`), [Table A.11](#) (`CustomTransferFunction`), and [Table A.12](#) (`RenderingIntent`).

Table A.6 — Customxy bundle

condition	type	default	name
	U32(Bits(19), BitsOffset(19, 524288), BitsOffset(20, 1048576), BitsOffset(21, 2097152))	0	ux
	U32(Bits(19), BitsOffset(19, 524288), BitsOffset(20, 1048576), BitsOffset(21, 2097152))	0	uy

$x = \text{UnpackSigned}(ux)$ and $y = \text{UnpackSigned}(uy)$ are the coordinates of a point on the CIE xy chromaticity diagram, scaled by 10^6 . The unscaled coordinates may be outside [0, 1] for imaginary primaries.

Table A.7 — ColourSpace

name	value	meaning
kRGB	0	Tristimulus RGB, with various white points and primaries
kGrey	1	Luminance, with various white points; <code>colour_encoding.primaries</code> are ignored
kXYB	2	XYB (opsin); <code>colour_encoding.white_point</code> is kD65, <code>colour_encoding.primaries</code> is ignored, <code>colour_encoding.have_gamma</code> is true, and <code>colour_encoding.gamma</code> is 3333333
kUnknown	3	None of the other table entries describe the colour space appropriately

Table A.8 — WhitePoint

name	value	meaning
kD65	1	CIE Standard Illuminant D65: 0.3127, 0.3290
kCustom	2	Custom white point stored in <code>colour_encoding.white</code>
kE	10	CIE Standard Illuminant E (equal-energy): 1/3, 1/3
kDCI	11	DCI-P3 from SMPTE RP 431-2: 0.314, 0.351

The `meaning` column is interpreted as CIE xy chromaticity coordinates.

NOTE 1 The values are a selection of the values defined in [2].

Table A.9 — Primaries

name	value	meaning
kSRGB	1	0.639998686, 0.330010138; 0.300003784, 0.600003357; 0.150002046, 0.059997204
kCustom	2	Custom red/green/blue primaries stored in <code>colour_encoding.red/green/blue</code>
k2100	9	0.708, 0.292; 0.170, 0.797; 0.131, 0.046 (As specified in Rec. ITU-R BT.2100-2)
kP3	11	0.680, 0.320; 0.265, 0.690; 0.150, 0.060 (As specified in SMPTE RP 431-2)

The values in the `meaning` column are interpreted as CIE xy chromaticity coordinates: red; green; blue, respectively.

NOTE 2 The values are a selection of the values defined in Rec. ITU-T H.273 | ISO/IEC 23091-2:2019 [3]. The xy coordinates of kSRGB are quantized and match the values that would be stored in an ICC profile.

Table A.10 — TransferFunction

name	value	meaning
k709	1	As specified in Rec. ITU-R BT.709-6
kUnknown	2	None of the other table entries describe the transfer function
kLinear	8	The gamma exponent is 1
kSRGB	13	As specified in IEC 61966-2-1 sRGB

Table A.10 (continued)

name	value	meaning
kPQ	16	As specified in Rec. ITU-R BT.2100-2 (PQ)
kDCI	17	As specified in SMPTE ST 428-1
kHLG	18	As specified in Rec. ITU-R BT.2100-2 (HLG)

NOTE 3 The values are a selection of the values defined in Rec. ITU-T H.273 | ISO/IEC 23091-2:2019 [3].

Table A.11 — CustomTransferFunction bundle

condition	type	default	name
	Bool()	false	have_gamma
have_gamma	u(24)	10 000 000	gamma
!have_gamma	Enum(TransferFunction)	TransferFunction.kSRGB	transfer_function

have_gamma == false indicates the transfer function is defined by transfer_function. Otherwise, the opto-electrical transfer function is characterized by the exponent $\gamma / 10^7$. This exponent is in (0, 1].

Table A.12 — RenderingIntent

name	value	meaning
kPerceptual	0	As specified in ISO 15076-1:2010 (vendor-specific)
kRelative	1	As specified in ISO 15076-1:2010 (media-relative)
kSaturation	2	As specified in ISO 15076-1:2010 (vendor-specific)
kAbsolute	3	As specified in ISO 15076-1:2010 (ICC-absolute)

NOTE 4 The values are defined by ISO 15076-1:2010.

Table A.13 — ColourEncoding bundle

condition	type	default	name
	Bool()	true	all_default
!all_default	Bool()	false	want_icc
!all_default	Enum(ColourSpace)	kRGB	colour_space
use_desc && not_xyb	Enum(WhitePoint)	kD65	white_point
white_point == WP.kCustom	Customxy		white
use_desc && not_xyb && colour_space != kGrey	Enum(Primaries)	PR.kSRGB	primaries
primaries == PR.kCustom	Customxy		red
primaries == PR.kCustom	Customxy		green
primaries == PR.kCustom	Customxy		blue
use_desc	CustomTransferFunction		tf
use_desc	Enum(RenderingIntent)	kRelative	rendering_intent

In Table A.13, CS denotes ColourSpace, WP denotes WhitePoint, PR denotes Primaries, use_desc expands to !all_default && !want_icc, and not_xyb expands to colour_space != kXYB.

want_icc is true if and only if the codestream stores an ICC profile. If so, it is decoded as specified in Annex B, and describes the colour space. Also, colour_space is kGrey if and only if the ICC profile is greyscale. Otherwise, there is no ICC profile and the other fields describe the colour space.

The applicability of ColourEncoding is specified in ImageMetadata (A.6) and Annex B.

A.5 Extensions

This bundle, specified in [Table A.14](#), is a field in bundles (ImageMetadata, FrameHeader, RestorationFilter) which can be extended (cf. [Annex H](#)).

Table A.14 — Extensions bundle

condition	type	default	name
	U64()	0	extensions
extensions != 0	U64()	0	extension_bits[NumExt]

An extension consists of zero or more bits whose interpretation is established by [Annex H](#). Each extension is identified by an `ext_i` in the range [0, 63], assigned in increasing sequential order. `extensions` is a bit array where the `i`-th bit (least-significant == 0) indicates whether the extension with `ext_i = i` is present. `NumExt` denotes the number of extensions present, i.e. number of 1-bits in `extensions`. Extensions that are present are stored in ascending order of their `ext_i`. `extension_bits[i]` indicates the number of bits stored for the extension whose `ext_i = i`, starting after `extension_bits` has been read. The decoder reads all these bits for all extensions which are present.

A.6 ImageMetadata

[Table A.15](#) specifies the BitDepth bundle.

Table A.15 — BitDepth bundle

condition	type	default	name
	Bool()	false	float_sample
!float_sample	U32(Val(8), Val(10), Val(12), BitsOffset(6,1))	8	bits_per_sample
float_sample	U32(Val(32), Val(16), Val(24), BitsOffset(6,1))	8	bits_per_sample
float_sample	u(4)		exp_bits_minus_one

`float_sample` is the data type of sample values (true for floating point, false for integers).

`bits_per_sample` is the number of bits per channel of the original image. The encoding of this value depends on `floating_point_sample`: for integers the value is in range [1, 31], for floating point restrictions depend on the value `exp_bits_minus_one`.

`exp_bits_minus_one` is one less than the amount of bits used for the exponent of floating point values and is only present if `float_sample` is true. Let `exp_bits = exp_bits_minus_one + 1` be the amount of exponent bits and let `mantissa_bits` be `bits_per_sample - exp_bits - 1`. The samples of the original image are interpreted as floating point values with 1 sign bit, the indicated amount of exponent bits and mantissa bits and otherwise following the principles of the ISO/IEC 60559 standard with an exponent bias of $2^{\text{exp_bits}-1} - 1$. The value of `exp_bits` is in the range [2, 8], and the value of `mantissa_bits` is in the range [2, 23].

If a sample's exponent has the highest representable value, the decoded value of this sample is unspecified and platform-dependent.

NOTE 1 On some CPUs and floating-point formats, infinity/Not A Number are disallowed and the largest possible exponent is treated in the same way as the next smaller one.

The ImageMetadata bundle, defined in [Table A.16](#), holds information that applies to all Frames (including the preview); decoders use it to interpret pixel values and restore the original image.

Table A.16 — ImageMetadata bundle

condition	type	default	name
	Bool()	true	all_default
!all_default	Bool()	false	extra_fields
extra_fields	u(3)	0	orientation_minus_1
extra_fields	Bool()	false	have_intr_size
have_intr_size	SizeHeader		intrinsic_size
extra_fields	Bool()	false	have_preview
have_preview	PreviewHeader		preview
extra_fields	Bool()	false	have_animation
have_animation	AnimationHeader		animation
!all_default	BitDepth		bit_depth
!all_default	Bool()	true	modular_16bit_buffers
!all_default	U32(Val(0), Val(1), BitsOffset(4, 2), BitsOffset(12, 1))	0	num_extra_channels
!all_default	ExtraChannelInfo		extra_channel_info [num_extra_channels]
!all_default	Bool()	true	xyb_encoded
!all_default	ColourEncoding		colour_encoding
extra_fields	ToneMapping		tone_mapping
!all_default	Extensions		extensions
	Bool()		default_transform
default_transform && xyb_encoded	OpsinInverseMatrix		opsin_inverse_matrix
default_transform	u(3)	0	cw_mask
cw_mask & 1	F16()	d_up2	up2_weight[15]
cw_mask & 2	F16()	d_up4	up4_weight[55]
cw_mask & 4	F16()	d_up8	up8_weight[210]

The `orientation = 1 + orientation_minus_1` indicates which orientation transform the decoder applies after decoding the image (Table A.17), including the preview frame if present.

Table A.17 — Orientation

orientation	side in first row	side in first column	transform to apply
1	top	left	none
2	top	right	flip horizontally
3	bottom	right	rotate 180°
4	bottom	left	flip vertically
5	left	top	transpose (rotate 90° clockwise then flip horizontally)
6	right	top	rotate 90° clockwise
7	right	bottom	flip horizontally then rotate 90° clockwise
8	left	bottom	rotate 90° counterclockwise

NOTE 2 orientation matches the values used by JEITA CP-3451C (Exif version 2.3).

If `have_intr_size`, then `intrinsic_size` denotes the recommended dimensions for displaying the image, i.e. applications are advised to resample the decoded image to the dimensions indicated in `intrinsic_size`.

`have_preview` indicates whether the codestream includes a `preview` and `preview_frame`.

`have_animation` indicates whether an AnimationHeader is included (`animation`) and whether FrameHeader (C.2) includes timing information.

`modular_16bit_buffers` indicates whether signed 16-bit integers have a large enough range to store and apply inverse transforms to all the decoded samples in modular image sub-bitstreams (see C.9). If false, 32-bit integers have to be used. In any case, signed 32-bit integers suffice to store decoded samples in modular image sub-bitstreams, as well as the results of inverse modular transforms. For some of the intermediate arithmetic (e.g. predictor computations), 64-bit arithmetic is needed.

`num_extra_channels` is the number of additional image channels. If it is nonzero, then `extra_channel_info` signals the semantics of each extra channel.

`xyb_encoded` is true if the stored image is in the XYB colour space. In L.2 it is specified how to convert XYB to RGB.

In the rest of this document, the three main colour components are referred to as X, Y, B, even in the case where the XYB colour space is not used for the stored image; in that case the actual components are either R, G, B or Cb, Y, Cr or C, M, Y.

`colour_encoding` indicates the colour image encoding of the original image. This may differ from the encoding of the stored image if `xyb_encoded` is true.

`tone_mapping` (Tables A.18) has information for mapping HDR images to lower dynamic range displays.

Table A.18 — ToneMapping bundle

condition	type	default	name
	Bool()	true	<code>all_default</code>
<code>!all_default</code>	F16()	255	<code>intensity_target</code>
<code>!all_default</code>	F16()	0	<code>min_nits</code>
<code>!all_default</code>	Bool()	false	<code>relative_to_max_display</code>
<code>!all_default</code>	F16()	0	<code>linear_below</code>

`intensity_target` is > 0 . This is an upper bound on the intensity level present in the image in nits, and represents the intensity corresponding to the value "1.0" (but the image need not contain a pixel this bright).

`min_nits` is > 0 and $\leq \text{intensity_target}$. This is a lower bound on the intensity level present in the image in nits, but the image need not contain a pixel this dark.

`linear_below` represents a value below which tone mapping leaves the values unchanged. If `relative_to_max_display` is true, `linear_below` is a ratio in $[0, 1]$ of the maximum display brightness in nits. Otherwise, it is an absolute brightness in nits ≥ 0 .

For the upsampling filters, custom weights are signalled depending on `cw_mask`. The default weights are defined as follows: $d_{up2} = \{-0.01716200, -0.03452303, -0.04022174, -0.02921014, -0.00624645, 0.14111091, 0.28896755, 0.00278718, -0.01610267, 0.56661550, 0.03777607, -0.01986694, -0.03144731, -0.01185068, -0.00213539\}$, $d_{up4} = \{-0.02419067, -0.03491987, -0.03693351, -0.03094285, -0.00529785, -0.01663432, -0.03556863, -0.03888905, -0.03516850, -0.00989469, 0.23651958, 0.33392945, -0.01073543, -0.01313181, -0.03556694, 0.13048175, 0.40103025, 0.03951150, -0.02077584, 0.46914198, -0.00209270, -0.01484589, -0.04064806, 0.18942530, 0.56279892, 0.06674400, -0.02335494, -0.03551682, -0.00754830, -0.02267919, -0.02363578, 0.00315804, -0.03399098, -0.01359519, -0.00091653, -0.00335467, -0.01163294, -0.01610294, -0.00974088, -0.00191622, -0.01095446, -0.03198464, -0.04455121, -0.02799790, -0.00645912, 0.06390599, 0.22963888, 0.00630981, -0.01897349, 0.67537268, 0.08483369, -0.02534994, -0.02205197, -0.01667999, -0.00384443\}$, and $d_{up8} = \{-0.02928613, -0.03706353, -0.03783812, -0.03324558, -0.00447632, -0.02519406, -0.03752601, -0.03901508, -0.03663285, -0.00646649, -0.02066407, -0.03838633, -0.04002101, -0.03900035, -0.00901973, -0.01626393, -0.03954148, -0.04046620, -0.03979621, -0.01224485, 0.29895328, 0.35757708, -0.02447552, -0.01081748, -0.04314594, 0.23903219, 0.41119301, -0.00573046, -0.01450239, -0.04246845, 0.17567618, 0.45220643, 0.02287757, -0.01936783, -0.03583255, 0.11572472, 0.47416733, 0.06284440, -0.02685066, 0.42720050, -0.02248939, -0.01155273, -0.04562755, 0.28689496, 0.49093869, -0.00007891, -0.01545926, -0.04562659, 0.21238920, 0.53980934, 0.03369474, -0.02070211, -0.03866988, 0.14229550, 0.56593398, 0.08045181, -0.02888298, -0.03680918, -0.00542229, -0.02920477, -0.02788574, -0.02118180, -0.03942402, -0.00775547, -0.02433614, -0.03193943, -0.02030828, -0.04044014, -0.01074016, -0.01930822, -0.03620399, -0.01974125, -0.03919545, -0.01456093, -0.00045072, -0.00360110, -0.01020207, -0.01231907, -0.00638988, -0.00071592, -0.00279122, -0.00957115, -0.01288327, -0.00730937, -0.00107783, -0.00210156, -0.00890705, -0.01317668, -0.00813895, -0.00153491, -0.02128481, -0.04173044, -0.04831487, -0.03293190, -0.00525260, -0.01720322, -0.04052736, -0.05045706, -0.03607317, -0.00738030, -0.01341764, -0.03965629, -0.05151616, -0.03814886, -0.01005819, 0.18968273, 0.33063684, -0.01300105, -0.01372950, -0.04017465, 0.13727832, 0.36402234, 0.01027890, -0.01832107, -0.03365072, 0.08734506, 0.38194295, 0.04338228, -0.02525993, 0.56408126, 0.00458352, -0.01648227, -0.04887868, 0.24585519, 0.62026135, 0.04314807, -0.02213737, -0.04158014, 0.16637289, 0.65027023, 0.09621636, -0.03101388, -0.04082742, -0.00904519, -0.02790922, -0.02117818, 0.00798662, -0.03995711, -0.01243427, -0.02231705, -0.02946266, 0.00992055, -0.03600283, -0.01684920, -0.00111684, -0.00411204, -0.01297130, -0.01723725, -0.01022545, -0.00165306, -0.00313110, -0.01218016, -0.01763266, -0.01125620, -0.00231663, -0.01374149, -0.03797620, -0.05142937, -0.03117307, -0.00581914, -0.01064003, -0.03608089, -0.05272168, -0.03375670, -0.00795586, 0.09628104, 0.27129991, -0.00353779, -0.01734151, -0.03153981, 0.05686230, 0.28500998, 0.02230594, -0.02374955, 0.68214326, 0.05018048, -0.02320852, -0.04383616, 0.18459474, 0.71517975, 0.10805613, -0.03263677, -0.03637639, -0.01394373, -0.02511203, -0.01728636, 0.05407331, -0.02867568, -0.01893131, -0.00240854, -0.00446511, -0.01636187, -0.02377053, -0.01522848, -0.00333334, -0.00819975, -0.02964169, -0.04499287, -0.02745350, -0.00612408, 0.02727416, 0.19446600, 0.00159832, -0.02232473, 0.74982506, 0.11452620, -0.03348048, -0.01605681, -0.02070339, -0.00458223\}$.

A.7 AnimationHeader

AnimationHeader, defined in [Table A.19](#), holds meta-information about an animation or image sequence that is present if `metadata.have_animation`.

Table A.19 — AnimationHeader bundle

condition	type	default	name
	U32(Val(100), Val(1000), BitsOffset(10, 1), BitsOffset(30, 1))	0	tps_numerator
	U32(Val(1), Val(1001), BitsOffset(8, 1), BitsOffset(10, 1))	0	tps_denominator
	U32(Val(0), Bits(3), Bits(16), Bits(32))	0	num_loops
	Bool()	false	have_timecodes

NOTE This document defines the interval between presenting the current and next frame in units of ticks. In the case where `metadata.have_animation` is false, the frames do not represent an animation, but layers that are overlaid to obtain a composite image.

`tps_numerator / tps_denominator` indicates the number of ticks per second.
`num_loops` is the number of times to repeat the animation (0 is interpreted as infinity).
`have_timecodes` indicates whether time codes are signalled in the frame headers.

A.8 OpsinInverseMatrix

OpsinInverseMatrix, defined in [Table A.20](#), specifies parameters related to the inverse XYZ colour transform.

Table A.20 — OpsinInverseMatrix bundle

condition	type	default	name
	Bool()	true	all_default
!all_default	F16()	11.031566901960783	inv_mat00
!all_default	F16()	-9.866943921568629	inv_mat01
!all_default	F16()	-0.16462299647058826	inv_mat02
!all_default	F16()	-3.254147380392157	inv_mat10
!all_default	F16()	4.418770392156863	inv_mat11
!all_default	F16()	-0.16462299647058826	inv_mat12
!all_default	F16()	-3.6588512862745097	inv_mat20
!all_default	F16()	2.7129230470588235	inv_mat21
!all_default	F16()	1.9459282392156863	inv_mat22
!all_default	F16()	-0.0037930732552754493	opsin_bias0
!all_default	F16()	-0.0037930732552754493	opsin_bias1
!all_default	F16()	-0.0037930732552754493	opsin_bias2
!all_default	F16()	1-0.05465007330715401	quant_bias0
!all_default	F16()	1-0.07005449891748593	quant_bias1
!all_default	F16()	1-0.049935103337343655	quant_bias2
!all_default	F16()	0.145	quant_bias_numerator

A.9 ExtraChannelInfo

[Table A.21](#) specifies the meaning of the ExtraChannelType values.

Table A.21 — ExtraChannelType

name	value	meaning
kAlpha	0	Alpha transparency, where 0 means fully transparent
kDepth	1	Depth map
kSpotColour	2	Spot colour channel; red, green, blue indicate its colour and solidity in [0, 1] indicates the overall blending factor, with 0 corresponding to fully translucent (invisible) and 1 corresponding to fully opaque.
kSelectionMask	3	Selection mask, which indicates a (fuzzy) region of interest, for example for image manipulation purposes. Pixels with value zero do not belong to the selection, pixels with the maximum value do belong to the selection.
kBlack	4	The K channel of a CMYK image. If present, a CMYK ICC profile is also present, and the RGB samples are to be interpreted as CMY, where 0 denotes full ink.
kCFA	5	Channel used to represent Colour Filter Array data (Bayer mosaic)
kThermal	6	Infrared thermography image. Sample values are in units of Kelvin.
kNonOptional	15	The decoder indicates it cannot safely interpret the semantics of this extra channel.

Table A.21 (continued)

name	value	meaning
kOptional	16	Extra channel that can be safely ignored.

Table A.22 specifies the ExtraChannelInfo bundle.

Table A.22 — ExtraChannelInfo bundle

condition	type	default	name
	Bool()	true	all_default
!all_default	ExtraChannelType	kAlpha	type
!all_default	BitDepth		bit_depth
!all_default	U32(Val(0), Val(3), Val(4), BitsOffset(3, 1))	0	dim_shift
!all_default	U32(Val(0), Bits(4), BitsOffset(5,16), BitsOffset(10,48))	0	name_len
	u(8)	0	name[name_len]
!all_default && type == kAlpha	Bool()	false	alpha_associated
type == kSpotColour	F16()	0	red
type == kSpotColour	F16()	0	green
type == kSpotColour	F16()	0	blue
type == kSpotColour	F16()	0	solidity
type == kCFA	U32(Val(1), Bits(2), BitsOffset(4, 3), BitsOffset(8, 19))	1	cfa_channel

Extra channels are interpreted according to their `type` and are rendered as specified in [L.6](#).

`dim_shift` is the base-2 logarithm of the downsampling factor of the dimensions of the extra channel with respect to the main image dimensions defined in `size`. The dimensions of the extra channel are rounded up. The value $1 \ll \text{dim_shift}$ does not exceed the `kGroupDim` of any frame ([C.2](#)).

EXAMPLE `dim_shift == 3` implies 8×8 downsampling, e.g. a 3×3 extra channel for a 20×20 main image.

If `name_len > 0`, then the extra channel has a name `name`, interpreted as a UTF-8 encoded string.

Annex B (normative)

ICC profile

B.1 General

The interpretation of RGB pixel data is governed by a colour space described by a colour profile, for example sRGB is a colour space. Capture devices, display devices, and quantized (integer) pixel data can all have their own colour space. Converting quantized pixels in one colour space to quantized pixels in a different colour space is a lossy operation.

Pixel data can be encoded in two ways: either in the absolute XYZ colour space, or as RGB triplets, YCbCr triplets or grayscale values which are to be interpreted according to the colour space described by a given colour profile.

NOTE 1 Both ways are useful in different scenarios:

- XYZ is designed to match the human visual system and is excellent for lossy compression.
- When performing lossless compression, integer data has to be kept in its original colour space, since converting it and re-quantizing it in another colour space is lossy.

A colour space is always signalled in the metadata, and it has a different meaning in each scenario:

- In the XYZ scenario, the signalled colour space does not give any information about how to interpret the encoded XYZ pixels, since they are already in an absolute colour space. It is merely intended to indicate what colour space the original pixel data had, which can be useful when converting the decoded image to integer data. To display the image data on a display device with a display profile, this is irrelevant since it suffices to convert from XYZ to the display colour space. The signalled colour space can be unused in that case. Only in case there are multiple frames that need to be blended by the decoder, the signalled colour space is relevant (since the blending may have to be done in that colour space and not in XYZ, see [C.2](#)).
- In the other scenario, the signalled colour space has direct meaning: it is the colour space of the RGB, YCbCr or grayscale pixel data. To display the image on a display device, the pixel data is to be converted from the signalled colour space to the display colour space.

The `metadata.xyb_encoded` flag ([A.6](#)) indicates which of the two scenarios is followed. In the case where `metadata.xyb_encoded` is true, `metadata.colour_encoding` and the ICC profile described in this Annex (if present) are merely suggestions as to a colour encoding to use after the intermediate conversion to the linear sRGB colour space specified in [L.2](#) (unless the frame is saved as a reference frame and `save_before_ct` is true, see [C.2](#)). If `xyb_encoded` is false, the decoded samples are to be interpreted in the following way:

- If `!metadata.colour_encoding.want_icc`, the decoded samples are indicated to be interpreted according to the `colour_space`, `white_point`, `primaries`, `transfer_function` and `rendering_intent` of `metadata.colour_encoding`, as specified in [A.4](#).
- Otherwise, the decoded samples are indicated to be interpreted according to an ICC profile decoded as specified in the remainder of this Annex.

EXAMPLE An original image could be in the DCI-P3 colour space. For lossless compression, an encoder could signal the DCI-P3 colour space using `metadata.colour_encoding`, set `metadata.xyb_encoded` to false, and encode the pixel values as they are. For lossy compression, an encoder could also signal the DCI-P3 colour space using `metadata.colour_encoding`, set `metadata.xyb_encoded` to true, and convert the pixel values to the XYZ colour space before encoding them. In this case, a decoder would decode the image in the XYZ colour space, and apply [L.2](#) to convert it to linear sRGB, which is an intermediate representation where some of the sample values could be negative or higher than the nominal maximum value since they are outside the sRGB gamut. To display the image on an sRGB display device, it would convert this intermediate representation to the sRGB colour space (i.e. apply clamping and adjust the transfer curve). To save the image in another (integer-based) image format, it could convert the intermediate representation to the suggested DCI-P3 colour space that was signalled, quantize the sample values to integers at the bit depth that was signalled, and save the resulting image.

The rest of this annex specifies how to decode an ICC profile.

NOTE 2 ICC profiles are standard display ICC profiles as defined by the International Colour Consortium. For example, an sRGB profile indicates the RGB values are to be interpreted as values in the sRGB colour space.

B.2 Data stream

The bitstream is byte aligned as specified in [6.3](#). The decoder reads `enc_size` as U64(). The decoder reads 41 clustered distributions as specified in [D.3](#). The decoder then reads `enc_size` integers as specified in [D.3.6](#) to obtain decompressed bytes, using `D[IccContext(index, prev_byte, prev_prev_byte)]` where `index` is the current byte index, `prev_byte` and `prev_prev_byte` are respectively the previous and second-previous bytes or 0 if they do not exist yet, and the `IccContext()` function is defined in the code below. The resulting decompressed byte-based data is the encoded ICC stream. The decoder reconstructs the ICC profile from this encoded ICC stream as described in the subsequent subclauses of this annex.

```
IccContext(i, b1, b2) {
    if (i <= 128) return 0;

    if (b1 >= 'a' && b1 <= 'z') p1 = 0;
    else if (b1 >= 'A' && b1 <= 'Z') p1 = 0;
    else if (b1 >= '0' && b1 <= '9') p1 = 1;
    else if (b1 == '.' || b1 == ',') p1 = 1;
    else if (b1 <= 1) p1 = 2 + b1;
    else if (b1 > 1 && b1 < 16) p1 = 4;
    else if (b1 > 240 && b1 < 255) p1 = 5;
    else if (b1 == 255) p1 = 6;
    else p1 = 7;

    if (b2 >= 'a' && b2 <= 'z') p2 = 0;
    else if (b2 >= 'A' && b2 <= 'Z') p2 = 0;
    else if (b2 >= '0' && b2 <= '9') p2 = 1;
    else if (b2 == '.' || b2 == ',') p2 = 1;
    else if (b2 < 16) p2 = 2;
    else if (b2 > 240) p2 = 3;
    else p2 = 4;

    return 1 + p1 + p2 * 8;
}
```

B.3 Encoded ICC stream

The encoded ICC stream is structured as shown in [Table B.1](#). In this table, Varint() and u() are read from the encoded ICC stream, not from the JPEG XL codestream.

Table B.1 — ICC stream

type	name	comment
Varint()	output_size	the size of the decoded ICC profile, in bytes
Varint()	commands_size	the size of the command stream, in bytes.
u(8 × commands_size)	command stream	sub-stream of the encoded ICC stream which contains command bytes
u(8 × [[remaining bytes]])	data stream	sub-stream of the encoded ICC stream which contains data bytes, starting directly after the command stream and ending at the end of the encoded ICC stream

The command stream is structured as shown in [Table B.2](#). In this table, u() is read from the command stream in the encoded ICC stream, not from the JPEG XL codestream.

Table B.2 — command stream

type	comment
u(8 × [[variable amount]])	commands for decoding tag list
u(8 × [[remaining bytes]])	commands for decoding main content

The data stream is structured as shown in [Table B.3](#). In this table, u() is read from the encoded data stream in the encoded ICC stream, not from the JPEG XL codestream.

Table B.3 — data stream

type	comment
u(8 × [[variable amount]])	data for decoding ICC header
u(8 × [[variable amount]])	data for decoding ICC tag list
u(8 × [[remaining bytes]])	data for decoding main content

The resulting ICC profile, output by the decoder, is structured as shown in [Table B.4](#). In this table, u() is read from the resulting ICC profile, not the JPEG XL codestream.

Table B.4 — ICC profile

type	name	comment
u(8 × [[variable amount]])	ICC header	up to 128 bytes long. Decoding procedure specified in B.4 .
u(8 × [[variable amount]])	ICC tag list	Decoding procedure specified in B.5 .
u(8 × [[remaining bytes]])	main content	Decoding procedure specified in B.6 .

After decoding the `output_size` and `commands_size`, the encoded ICC stream is considered to be divided into two input streams: the command stream, which starts at the current position and runs for `commands_size` bytes, and the data stream, which starts directly after the command stream and contains all remaining bytes. The decoder maintains the current positions within the command stream and the data stream, initially at the beginning of each respective stream, and increment each when a byte from the corresponding stream is read as indicated in the next subclauses. A stream position does not go beyond the last byte of that stream, and the commands stream does not extend beyond the end of the encoded ICC stream.