International Standard

ISO/IEC 18181-2

Second edition
2024-06

Information technology — JPEG XL image coding system —

Part 2: File format

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

Partie 2: Format de fichiers

Reference number
ISO/IEC 18181-2:2024(en)

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

Published in Switzerland

Contents

Page

Foreword

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

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

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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

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

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

This second edition cancels and replaces the first edition (ISO/IEC 18181-2:2021), which has been technically revised.

The main changes are as follows:

— Cross-references to ISO/IEC 18181-1 are updated to match its second edition;

— The JPEG bitstream reconstruction procedure was moved to Annex A and revised to improve clarity;

— Annex B was added, specifying the `image/jxl` media type registration.

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.

Information technology — JPEG XL image coding system —

Part 2:
File format

1 Scope

This document specifies the transport and container formats for JPEG XL codestreams as specified in ISO/IEC 18181-1. This document specifies how to add metadata and extensions to JPEG XL codestreams. A file as described by this document is called a JPEG XL file.

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/IEC 18181-1:2024, *Information technology — JPEG XL Image Coding System — Part 1: Core coding system*

ISO/IEC 10918-1:1994, *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*

ISO/IEC 19566-5, *Information technologies — JPEG systems — Part 5: JPEG universal metadata box format (JUMBF)*

IETF RFC 7932, *Brotli Compressed Data Format*[1)]

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
box
structured collection of data describing the image or the image decoding process

3.2
box content
data wrapped within the box structure

3.3
box type
kind of information stored within the box

3.4
file format
set of data structures for the storage of metadata and extensions of a codestream

1) https://www.rfc-editor.org/info/rfc7932

3.5
JPEG XL file
data file encoded in the file format defined by this document

3.6
superbox
box that carries other boxes as payload data

4 General

This document defines the file format of a JPEG XL file.

A JPEG XL file shall contain a codestream as specified in ISO/IEC 18181-1 and may contain additional metadata and extensions.

A JPEG XL file shall come in one of the following forms:

— A box structure, as defined in Clause 5;

— A direct JPEG XL codestream without box structure.

The rest of this document only defines the box structure, the codestream without box structure is valid but is completely specified in ISO/IEC 18181-1.

A decoder shall require the file format to follow either the structure of a codestream without box structure, or follow the box structure as defined in Clause 5 and follow all box requirements in Clauses 6 to 9. A decoder can extract the codestream from the box structure and decode the image from the codestream using the procedure specified in ISO/IEC 18181-1 and can decode the contents of other boxes following their respective specifications in this document.

NOTE A direct JPEG XL codestream without box structure is also a valid JPEG XL file. This allows, for example, a more efficient encoding of images for the web, in cases where information encoded in other boxes than the codestream is not required.

The JPEG XL media type registration for `image/jxl` is specified in Annex B.

5 File organization

A JPEG XL file using the box structure is formed as a series of boxes. These boxes contain all data within the file, including the initial signature required by the file format.

NOTE This box-based file format is based on the same syntax as described in ISO/IEC 15444-1:2019, Annex I or ISO/IEC 15444-2:2021, Annex M, or ISO/IEC 21122-3. The binary format of a box is also described in Clause 8.

Boxes of different types contain different types of data, such as the file signature, metadata and the codestream. Clause 9 defines box types that may appear in a JPEG XL file and their requirements. Boxes with an unrecognized type shall be ignored and skipped by the decoder.

A JPEG XL file shall contain a JPEG XL codestream. The codestream can be split across multiple boxes: JPEG XL partial codestream boxes. In this case, the codestream is formed by the concatenation of the content of all those boxes.

Any boxes, content and codestreams present in a superbox, such as another JPEG XL file in a JUMBF superbox, shall not be taken into account for the syntactic requirements of this document; they recursively follow their applicable specification.

Tables 1 and 2 each show a conceptual box structure of a JPEG XL file, that is a possible series of different box types that form the file, respectively with a single full codestream box and with multiple partial codestream boxes. Boxes that may appear multiple times are indicated with '…', optional boxes are indicated with brackets and required boxes are indicated in **bold**. These figures are only an indication and do not imply

any ordering or counting requirements for the boxes. The decoder shall not make any assumptions about the ordering of any boxes after the first two, except where indicated.

Table 1 — Conceptual structure (example) of a JPEG XL file using a full codestream box

JPEG XL Signature box
File Type box
Level box
(JPEG XL Frame Index box)
JPEG XL Codestream box
(JUMBF box) …
(Brotli-compressed box containing Exif)
(XML box) …
(Brotli-compressed box containing XML)
(JPEG Bitstream Reconstruction box)

Table 2 — Conceptual structure (example) of a JPEG XL file using partial codestream boxes

JPEG XL Signature box
File Type box
JPEG XL Partial Codestream box …
(JPEG XL Frame Index box)
JPEG XL Partial Codestream box …
(JUMBF box) …
(Exif box)
JPEG XL Partial Codestream box …
(XML box) …

6 Data types and numerical values

Data types used in this document shall be interpreted by the decoder as follows:

— u32: a 32-bit unsigned integer encoded in big endian order (4 bytes).

— u64: a 64-bit unsigned integer encoded in big endian order (8 bytes).

— Varint(): an unsigned integer value of up to 63 bits as a variable length integer in little endian order as specified in ISO/IEC 18181-1:2024, E.4.2.

— U32(), u(n), Bool: as specified in ISO/IEC 18181-1:2024, B.2.

Numerical values for bytes are given as hexadecimal values, each individually prefixed by 0x. Hexadecimal byte values are given in the order as they appear in the file. In some cases, these bytes spell out text in ASCII, this is informatively indicated after the hexadecimal values.

7 Graphical descriptions

Box definitions contain graphical description tables to illustrate the structure of the box. These tables should be interpreted as follows.

— A sequence of columns is used to indicate the fields of the box and their order (from left to right).

— Optional fields are indicated with brackets.

— Underline indicates a variable length field. Exact data types or sizes are indicated by name either in the rectangle after the name of the field, or in a description of the fields outside of the table.

— Multi-column headers may show fields that are grouped in a larger named structure.

Table 3 shows an example of a box with 3 fields:

— A: a name given to a group of the three fields contained within.

— B: required field with a fixed length data type: the type u32

— C: optional field with a fixed length data type (e.g. u32, u64 or a fixed amount of bytes)

— D: required field with a variable length data type (such as Varint(), or remaining amount of bytes)

Table 3 — Example of a graphical description of a box definition

A		
B: u32	(C)	D

8 Binary format of a box

Each box shall have the structure indicated in Table 4. This structure consists of a header indicating size and box type, and box content.

NOTE 1 This format is also specified in ISO/IEC 15444-1:2019 and ISO/IEC 15444-2:2021.

Table 4 — Binary format of a box

Box header			Box content
LBox: u32	TBox: 4 bytes	(XLBox: u64)	DBox: remaining bytes

The fields given in Table 4 are the following:

— LBox: has type u32. Gives the size of the box in bytes, including the box header fields. If the value is 1, then XLBox is used instead to indicate the size of the box. If the value is 0, then this box is the last box of the file, and its data extends to the end of the file. If the value is not 0 or 1, it shall be at least 8.

— TBox: has 4 bytes (e.g. a FourCC code): box type, specifies the type of information found in the box content, e.g. whether it is a JPEG XL Signature box, a File Type box, and so on.

— XLBox: has type u64. Only present if LBox == 1. If present this field, instead of the LBox field, indicates the size of the box in bytes. Its value shall be at least 16.

— DBox: has the remaining bytes. The box content (data). The content is formed by all the remaining bytes of the box. The size of the content in bytes is the box size minus the size of the box header fields. The format and meaning of this content is indicated by the box type, and Clause 9 defines the format of the contents that may appear in a JPEG XL file.

NOTE 2 The box size is a multiple of bytes. This includes the JPEG XL codestream box. The JPEG XL codestream is zero-padded at the end to align to a byte.

9 Box types

9.1 JPEG XL Signature box

The JPEG XL signature box shall contain exactly the following 12 bytes, given as hexadecimal numbers:

— `0x00 0x00 0x00 0x0C`

— `0x4A 0x58 0x4C 0x20` (the box type `JXL ` in ASCII)

— `0x0D 0x0A 0x87 0x0A`

A JPEG XL file shall contain exactly one signature box. The signature box shall be the first box.

9.2 File Type box

The file type box shall contain exactly the following 20 bytes:

— `0x00 0x00 0x00 0x14`

— `0x66 0x74 0x79 0x70` (the box type `ftyp` in ASCII)

— `0x6A 0x78 0x6C 0x20` (`jxl ` in ASCII)

— `0x00 0x00 0x00 0x00`

— `0x6A 0x78 0x6C 0x20` (`jxl ` in ASCII)

A JPEG XL file shall contain exactly one file type box. The file type box shall be the second box. The profile of the codestream box contained in this file is the Main profile.

9.3 Level box

The type of this box shall be given by the 4 bytes `0x6A 0x78 0x6C 0x6C` (`jxll` in ASCII).

Table 5 shows the contents of a Level box, excluding the box header.

Table 5 — Content of a Level box

level: u8

A JPEG XL file shall contain at most one Level box. If it is present, it shall be the third box, immediately after the file type box.

If there is no Level box, the level is assumed to be 5. This level applies to the content of the JPEG XL codestream box, as described in ISO/IEC 18181-1:2024, Annex M.

9.4 JUMBF box

The type of this box shall be given by the 4 bytes `0x6A 0x75 0x6D 0x62` (`jumb` in ASCII). This box shall follow the specification defined by ISO/IEC 19566-5.

A JUMBF box is a superbox that shall contain exactly one JUMBF Description box followed by one or more Content Boxes.

9.5 Exif box

The type of this box shall be given by the 4 bytes `0x45 0x78 0x69 0x66` (`Exif` in ASCII).

Table 6 shows the contents of an Exif box, excluding the box header.

Table 6 — Content of an Exif box

tiff header offset: u32	Exif payload: remaining bytes

The Exif payload is as described in JEITA CP-3451E or JEITA CP-3461B. The tiff header offset denotes, as specified in JEITA CP-3461B, the number of bytes, counting from the first byte of the Exif payload to the first

byte of the TIFF header of the Exif metadata. The value is zero if the payload starts immediately with the TIFF header.

NOTE 1 The content of this box is exactly ExifDataBlock as defined in ISO/IEC 23008-12:2022, Annex A.2.

For any Exif fields that have equivalents within the codestream, a decoder shall consider the codestream to take precedence. Encoders are encouraged to ensure the Exif and codestream fields are identical.

NOTE 2 Examples of such fields include orientation and pixel dimensions.

9.6 XML box

The type of this box shall be given by the 4 bytes `0x78 0x6D 0x6C 0x20` (`xml ` in ASCII).

The XML box contains a well-formed XML document as defined by W3C REC-xml-20081126.

Table 7 shows the content of an XML box.

Table 7 — Content of an XML box

XML data: all bytes

A JPEG XL file may contain multiple XML boxes.

NOTE This box follows the specification of XML Box in ISO/IEC 15444-2:2021.

9.7 Brotli-compressed box

The type of this box shall be given by the 4 bytes `0x62 0x72 0x6F 0x62` (`brob` in ASCII).

Table 8 shows the contents of a Brotli-compressed box, excluding the box header.

Table 8 — Content of a Brotli-compressed box

payload box type: 4 bytes	Brotli-compressed payload: remaining bytes

This box shall be treated as if it is a box of the type given by the first 4 bytes of its contents (the payload box type), with a contents equal to the Brotli-decompressed data obtained from the remaining bytes. The payload box type shall not be `0x62 0x72 0x6F 0x62` (`brob`) and shall not start with `0x6A 0x78 0x6C` (`jxl`) nor be equal to `0x6A 0x62 0x72 0x64` (`jbrd`).

Brotli-compressed data shall be decoded as specified in IETF RFC 7932.

9.8 Frame Index box

The type of this box shall be given by the 4 bytes `0x6A 0x78 0x6C 0x69` (`jxli` in ASCII).

This box contains an index of animation frame offsets. This box is optional and allows a decoder to efficiently seek the data of a frame based on time or frame order. Not all frames are necessarily listed in the index. All frames listed in the index shall be “keyframes”. The first frame shall always be listed. Keyframes are defined such that when a decoder seeks to the beginning of this frame, the result of decoding this frame and future frames is the same as when the decoder starts from the beginning. This implies that the current frame does not use earlier frames for features such as blending, patches or `lf_level`, and later frames can only refer to this frame or later for these features. The JPEG XL codestream supports frames with a duration of 0 ticks. These frames are not presented by the decoder but can be used to form composite frames such as through blending. Such frames are not counted as frames for the purpose of the F_i fields listed in Table 9, but the offset can point to such frames, as they are required for decoding the full composite frame and may form the beginning of a composite keyframe.

The box content shall have the structure indicated in Table 9 and further described below. In this table, all fields have type Varint() unless indicated otherwise.

Table 9 — Content of a Frame Index box

NF	T_{NUM}: u32	T_{DEN}: u32	OFF_0	T_0	F_0	...	OF_{FNF-1}	T_{NF-1}#	F_{NF-1}

The fields in Table 9 shall be interpreted as follows:

— NF: has type Varint(): number of frames listed in the index.

— T_{NUM}: has type u32: numerator of tick unit.

— T_{DEN}: has type u32: denominator of tick unit. If this value is 0, the file is ill-formed.

— per frame i listed:

  — OFF_i: has type Varint(): offset of start byte of this frame compared to start byte of previous frame from this index in the JPEG XL codestream. For the first frame, this is the offset from the first byte of the JPEG XL codestream.

  — T_i: has type Varint(): duration in ticks between the start of this frame and the start of the next frame in the index. If this is the last frame in the index, this is the duration in ticks between the start of this frame and the end of the stream. A tick lasts T_{NUM} / T_{DEN} seconds.

  — F_i: has type Varint(): amount of frames the next frame in the index occurs after this frame. If this is the last frame in the index, this is the amount of frames after this frame in the remainder of the stream. Only frames that are presented by the decoder are counted for this purpose, this excludes frames that are not intended for display but for compositing with other frames, such as frames that aren't the last frame with a duration of 0 ticks.

It is the responsibility of the creator of the file to ensure the frame index box corresponds to the codestream, in particular that the offsets point to the correct location of the corresponding frame, the frames are keyframes and the first frame is present in the list.

There shall be either zero or one Frame Index boxes in a JPEG XL file.

The Frame Index box may come before or after partial codestream boxes. Encoders are encouraged to write the frame index before the codestream, but the codestream header before the frame index, if possible.

NOTE 1 If the Frame Index box appears before the frames of the codestream, a streaming decoder would be able to issue range requests for an individual frame. If a partial codestream precedes the frame index box, it would preferably be small, for example containing only the codestream header and a preview image, which is useful to get image dimensions and preview from the earliest bytes of the file.

NOTE 2 The offsets OFF_i per frame are given as bytes in the codestream, not as bytes in the file format using the box structure. This means if JPEG XL Partial Codestream boxes (as defined in subclause 9.10) are used, the offset is counted within the concatenated codestream, bytes from box headers or non-codestream boxes are not counted.

9.9 JPEG XL Codestream box

The type of this box shall be given by the 4 bytes `0x6A 0x78 0x6C 0x63` (`jxlc` in ASCII). Its contents shall be the codestream described in ISO/IEC 18181-1.

A JPEG XL file shall contain either exactly one JPEG XL codestream box, or one or more JPEG XL partial codestream boxes, but not both.

Encoders are encouraged to place any metadata that might affect rendering before the partial or full codestream.

9.10 JPEG XL Partial Codestream box

The type of this box shall be given by the 4 bytes `0x6A 0x78 0x6C 0x70` (`jxlp` in ASCII). Its contents shall contain an index and a part of the codestream described in ISO/IEC 18181-1. Table 10 shows the contents of a JPEG XL Partial Codestream box, excluding the box header.

Table 10 — Content of a JPEG XL Partial Codestream box

index: u32	Partial codestream payload: remaining bytes

This type of box allows to split the codestream into multiple parts. When partial codestream boxes are used, the full codestream is formed by the concatenation of the partial codestream payload of all partial codestream boxes in order of increasing index. The index modulo 2^{31} shall be 0 for the first partial codestream box, and incremented by 1 for each next partial codestream box. The index shall be lower than 2^{31}, except for the last partial codestream box, which shall have an index of at least 2^{31}. The boxes shall appear in the file in order of increasing index. The full concatenation of all partial codestream boxes in this order shall form exactly one complete and valid JPEG XL codestream.

A JPEG XL file shall contain either exactly one JPEG XL codestream box, or one or more JPEG XL partial codestream boxes, but not both.

Encoders are encouraged to place any metadata that might affect rendering before the partial or full codestream.

NOTE 1 Splitting the codestream into multiple parts allows placing the first part of the codestream (e.g. the header and preview) early in the file, followed by possible metadata boxes, followed by the main part of the codestream.

NOTE 2 Partial codestream boxes can have an empty payload.

9.11 JPEG Bitstream Reconstruction Data box

The type of this box shall be given by the 4 bytes `0x6A 0x62 0x72 0x64` (`jbrd` in ASCII).

This subsection uses the same notation conventions as in ISO/IEC 18181-1. The decoder reads the fields listed in Table 11 (which refers to bundles defined in Tables 12 to 18).

Table 11 — JPEGBitstream bundle

condition	type	default	name
	Bool()		`is_grey`
	`0xC0` + Bits(6)		`marker[0]`
`for(i = 1; marker[i-1] != 0xD9; i++)`	`0xC0` + Bits(6)		`marker[i]`
	AppMarker		`app_marker[num_app_markers]`
	1 + Bits(16)		`com_length[num_com_markers]`
	1 + Bits(2)		`num_quant_tables`
	QuantTable		`quant[num_quant_tables]`
	Bits(2)		`comp_type`
`comp_type == 3`	1 + Bits(2)	`numc`	`num_comp`
`comp_type == 3`	Bits(8)	`d_id`	`component_id[num_comp]`
	Bits(2)		`component_q_idx[num_comp]`
	U32(Val(4), BitsOffset(3, 2), BitsOffset(4, 10), BitsOffset(6, 26))		`num_huff`
	HuffmanCode		`huffman_code[num_huff]`
	ScanInfo		`scan_info[num_scans]`
`has_dri`	Bits(16)	0	`restart_interval`
	ScanMoreInfo		`scan_more_info[num_scans]`
	Bits(16)		`intermarker_length[num_intermarker]`

Table 11 *(continued)*

condition	type	default	name
	U32(Val(0), BitsOffset(8, 1), BitsOffset(16, 257), BitsOffset(22, 65793))		`tail_data_length`
	Bool()		`has_padding`
`has_padding`	Bits(24)	0	`nbit`
	Bool()		`bbit[nbit]`

Here `marker` is an array of integers, and `num_app_markers` is the number of occurences of a number between `0xE0` and `0xEF` in this array, `num_com_markers` is the number of occurrences of the number `0xFE`, `num_scans` is the number of occurrences of the number `0xDA`, `num_intermarker` is the number of occurrences of the number `0xFF`, and `has_dri` is true if the number `0xDD` occurs.

The default number of components `numc` is equal to 1 if `comp_type == 0` and 3 otherwise. The default component identifiers `d_id` is equal to {1} if `comp_type == 0`, to {1,2,3} if `comp_type == 1`, and to {'R', 'G', 'B'} if `comp_type == 2`.

Table 12 — AppMarker bundle

condition	type	default	name
	U32(Val(0), Val(1), BitsOffset(1, 2), BitsOffset(2, 4))		`type`
	1 + Bits(16)		`length`

Table 13 — QuantTable bundle

condition	type	default	name
	Bits(1)		`precision`
	Bits(2)		`index`
	Bool()		`is_last`

Table 14 — HuffmanCode bundle

condition	type	default	name
	Bool()		`is_ac`
	Bits(2)		`id`
	Bool()		`is_last`
	U32(Val(0), Val(1), BitsOffset(3, 2), Bits(8))		`counts[16]`
	U32(Bits(2), BitsOffset(2, 4), BitsOffset(4, 8), BitsOffset(8, 1))		`values[sum(counts)]`

Table 15 — ScanInfo bundle

condition	type	default	name
	1 + Bits(2)		`num_comps`
	Bits(6)		`Ss`
	Bits(6)		`Se`
	Bits(4)		`Al`
	Bits(4)		`Ah`
	ScanComponentInfo		`component[num_comp]`
	U32(Val(0), Val(1), Val(2), BitsOffset(3, 3))		`last_needed_pass`

Table 16 — ScanComponentInfo bundle

condition	type	default	name
	Bits(2)		`comp_idx`
	Bits(2)		`ac_tbl_idx`
	Bits(2)		`dc_tbl_idx`

Table 17 — ScanMoreInfo bundle

condition	type	default	name
	U32(Val(0), BitsOffset(2, 1), BitsOffset(4, 4), BitsOffset(16, 20))		`num_reset_points`
	U32(Val(0), BitsOffset(3, 1), BitsOffset(5, 9), BitsOffset(28, 41))		`reset_point[num_reset_points]`
	U32(Val(0), BitsOffset(2, 1), BitsOffset(4, 4), BitsOffset(16, 20))		`num_extra_zero_runs`
	ExtraZeroRun		`extra_zero_run[num_extra_zero_runs]`

Table 18 — ExtraZeroRun bundle

condition	type	default	name
	U32(Val(1), BitsOffset(2, 2), BitsOffset(4, 5), BitsOffset(8, 20))		`num_runs`
	U32(Val(0), BitsOffset(3, 1), BitsOffset(5, 9), BitsOffset(28, 41))		`run_length`

After reading these fields, the decoder decompresses a single Brotli stream as specified in IETF RFC 7932. The decompressed stream contains a concatenation of the following data:

— First the contents of every unknown app marker: for `i` from 0 to `num_app_markers - 1`, if `app_marker[i].type` is 0, then `app_marker[i].length` bytes are given corresponding to APPn segment `app_data[i]` (otherwise the bytes will be provided in a different way);

— Then for `i` from 0 to `num_com_markers - 1`, there are `com_length[i]` bytes corresponding to COM segment `com_data[i]`;

— Then for `i` from 0 to `num_intermarker - 1`, there are `intermarker_length[i]` bytes corresponding to unrecognized segment `intermarker_data[i]`;

— Finally, there are `tail_data_length` bytes which denoted as `tail_data`.

The procedure to reconstruct a JPEG bitstream based on the data contained in the JPEG Bitstream Reconstruction Data box (as well as the JPEG XL Codestream box and potentially other boxes) is specified in Annex A.

Annex A
(normative)

JPEG Bitstream Reconstruction procedure

A.1 General

The reconstructed JPEG bitstream is produced as a concatenation of {`0xFF`, `0xD8`} bytes (implicit SOI segment) and a sequence of segments, generated on the basis of the elements of the `marker` array. Element values determine the type of the corresponding segment. Detailed instructions for various type values are specified in subclauses. If the encountered type does not match any instruction, then the codestream is ill-formed.

`app_data`, `com_data`, `inter_marker_data`, `huffman_code`, `quant`, `bbit` and `scan_info` members are accessed through "iterators"; this means that during JPEG reconstruction every element (referenced as "*next*") is used exactly once and in order of increasing index.

A.2 SOF segment

"Start Of Frame" segment.

`type` is one of {`0xC0`, `0xC1`, `0xC2`, `0xC9`, `0xCA`}.

This segment starts with 10 bytes: {`0xFF`, `type`, `len_hi`, `len_lo`, 8, `height_hi`, `height_lo`, `width_hi`, `width_lo`, `num_comp`}, where `_hi` and `_lo` denote the highest and lowest 8 bits of the corresponding values, `len` is the final length of this segment minus 2, and `height` and `width` are the image dimensions (see D.2.

For each of the `num_comp` components, 3 more bytes are appended to this segment: {`component_id[i]`, `(h_samp_factor[i] << 4) | v_samp_factor[i]`, `component_q_idx[i]`}, where `h_samp_factor` (`v_samp_factor`) is the horizontal (vertical) sampling factor of the corresponding component according to the `jpeg_upsampling` field (see F.2) — note that in case of YCbCr, the order is Cb, Y, Cr in the `jpeg_upsampling` field while it is Y, Cb, Cr in the reconstructed JPEG bitstream.

A.3 DHT segment

"Define Huffman Table(s)" segment.

`type` is `0xC4`.

In this segment a series of HuffmanCode entities are serialized. Entities are fetched from the *next* `huffman_code` until the element with `is_last == true` is reached.

This segment starts with 4 bytes: {`0xFF`, `0xC4`, `len_hi`, `len_lo`}, where `len_hi` and `len_lo` are the highest and lowest 8 bits of the final length of this segment minus 2.

For each HuffmanCode entity `hc`, the segment content is defined in ISO/IEC 10918-1:1994, Annex B.2.4.2, with the following mapping:

— Tc together with Th, encoded as a single byte, is `hc.id`

— L_i are corresponding `hc.counts` elements, except for the last non-zero element, which shall be decremented by 1, before storing

— (flattened) $V_{i,j}$ are corresponding `hc.values` elements

A.4 RSTn segment

"Restart" segment.

`type` is in the range [`0xD0`, `0xD8`).

This segment contains 2 bytes: {`0xFF`, `type`}.

A.5 EOI segment

"End Of Image" segment; this is the last segment.

`type` is `0xD9`.

This segment starts with {`0xFF`, `0xD9`} bytes. The rest of the segment is copied from `tail_data`.

A.6 SOS segment

"Start Of Scan" segment.

`type` is `0xDA`.

Let `si` be the *next* `scan_info` element, `smi` be the *next* `scan_more_info` element, `num_comps = si.num_comps`, and `len = 6 + 2 * num_comps`.

This segment starts with {`0xFF`, `0xDA`, `len_hi`, `len_lo`, `num_comps`} bytes, where `len_hi` and `len_lo` are the highest and lowest 8 bits of `len`.

For each `si.component` element `csi`, the segment content is defined in ISO/IEC 10918-1:1994, Annex B.2.3, with the following mapping:

— Cs_j is `component_id[csi.comp_idx]`

— Td_j is `csi.dc_tbl_idx`

— Ta_j is `csi.ac_tbl_idx`

The next 3 bytes of this segment are also defined in ISO/IEC 10918-1:1994, Annex B.2.3, with the following mapping: Ss is `si.Ss`, Se is `si.Se`, Ah is `si.Ah`, and Al is `si.Al`.

The DCT coefficient data is encoded according to ISO/IEC 10918-1:1994 with the following changes to enable unambiguous bit-precise JPEG stream reconstruction:

— if `has_padding` then making entropy-coded segments (ISO/IEC 10918-1:1994, Annex B.1.1.5) an integer number of bytes is performed as follows: *next* bits from `bbit` are used, if necessary, to pad to the end of the compressed data to complete the final byte of a segment (otherwise these padding bits are zero);

— when encoding AC coefficients for sequential DCT (ISO/IEC 10918-1:1994, Annex F.1.2.2.1), `ezr.num_runs` extra "ZRL" symbols are emitted before processing the end-of-block, where `ezr` is the `smi.extra_zero_run` element whose `block_idx` member matches the currently serialized block index (in the current scan); the final "EOB" symbol is emitted only if the number of 0 coefficients remaining to be encoded is non-negative;

— when encoding AC coefficients for progressive DCT (ISO/IEC 10918-1:1994, Annex G.1.2.2), `ezr.num_runs` extra "ZRL" symbols are emitted before updating "EOBRUN", where `ezr` is the `smi.extra_zero_run element` whose `block_idx` member matches the currently serialized block index (in the current scan); "EOBRUN" is updated only if the number of 0 coefficients remaining to be encoded is non-negative

— before encoding the block, if the block with the currently serialized block index (in the current scan) is present in the `smi.reset_poin`t set, then "Encode_EOBRUN" is invoked (ISO/IEC 10918-1:1994, Annex G.1.2.2).

A.7 DQT segment

“Define Quantization Table(s)” segment.

`type` is `0xDB`.

In this segment a series of QuantTable entities are serialized. Series of items are fetched from the *next* `quant` until the element with `is_last == true` is reached.

This segment starts with {`0xFF`, `0xDB`, `len_hi`, `len_lo`} bytes, where `len_hi` and `len_lo` are the highest and lowest 8 bits of the final length of this segment minus 2.

For each QuantTable element `q`, the serialized content is defined in ISO/IEC 10918-1:1994, Annex B.2.4.1, with the following mapping:

— Pq is `q.precision`

— Tq is `q.index`

— Q_k are corresponding quantization factors from the JPEG XL codestream (I.2.4). If there are no corresponding quantization factors (which implies that this is an unused quantization table), the factors are to be considered identical to those of the previous QuantTable element.

A.8 DRI segment

“Define Restart Interval” segment.

`type` is `0xDD`.

This segment contains {`0xFF`, `0xDD`, `0x00`, `0x04`, `hi`, `lo`} bytes, where `hi` and `lo` are the highest and lowest 8 bits of `restart_interval`.

A.9 APPn segment

“Application-specific” segment.

`type` is in the range [`0xE0`, `0xF0`).

This segment starts with the {`0xFF`} byte. Let `am` be the *next* `app_marker`. If `am.type` is 0, then the rest of the segment is copied from the corresponding `app_data` element. If it has a different type, then this segment is constructed as follows:

If `am.type` is 1 (ICC profile), then the next bytes are {`0xE2`, `len_hi`, `len_lo`}, where `len_hi` and `len_lo` are the highest and lowest 8 bits of `am.length - 1`, followed by the zero-terminated ASCII string “ICC_PROFILE”, followed by the u(8) index among all app markers of type 1 (counting from 1), followed by the u(8) total number of app markers of type 1, followed by the *next* fragment of the decoded ICC profile (as described in E.4) of length `am.length - 17`.

If am.type is 2 (Exif metadata) or 3 (XMP metadata), then the next bytes are {`0xE1`, `len_hi`, `len_lo`}, where `len_hi` and `len_lo` are the highest and lowest 8 bits of `am.length - 1`. Then if the type is 2, this is followed by the zero-terminated string “Exif”, followed by another zero, followed by the payload of the (next) Exif box (9.5) not including the tiff header offset, or a Brotli-compressed equivalent (9.7). If the type is 3, then this is followed by the the zero-terminated string “http://ns.adobe.com/xap/1.0/”, followed by the payload of the (next) XML box (9.6) or a Brotli-compressed equivalent (9.7).

A.10 COM segment

“Comment” segment.

`type` is `0xFE`.