INFORMATION TECHNOLOGY

JPEG 2000 IMAGE CODING SYSTEM

JPEG 2000 COMMITTEE DRAFT VERSION 1.0, 9 DECEMBER 1999

THE ISO AND ITU WILL PROVIDE COVER PAGES.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Foreword

Forward to be supplied by ISO and ITU.

Introduction

Introduction to be supplied by ISO and ITU.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY –
JPEG 2000 IMAGE CODING SYSTEM

1 Scope

This Recommendation | International Standard defines a set of lossless (bit-preserving) and lossy compression methods for coding continuous-tone, bi-level, grey-scale, or colour digital still images.

This Recommendation | International Standard

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

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

— specifies a file format

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

— provides guidance on how to implement these processes in practice

2 References

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

— ITU-T Recommendation T.81 | ISO/IEC 10918-1:1994, Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines.

— ITU-T Recommendation T.82 | ISO/IEC 11544:1994, Information technology - Coded representation of picture and audio information — Progressive bi-level image compression

— ITU-T Recommendation T.83 | ISO/IEC 10918-2:1995, Information technology - Digital compression and coding of continuous-tone still images: Compliance testing.

— ITU-T Recommendation T.84 | ISO/IEC 10918-3:1996, Information technology - Digital compression and coding of continuous-tone still images: Extensions.

— ITU-T Recommendation T.84 | ISO/IEC 10918-3 Amd 1 (In preparation), Information technology - Digital compression and coding of continuous-tone still images: Extensions - Amendment 1.

— ITU-T Recommendation T.86 | ISO/IEC 10918-4, Information technology - Digital compression and coding of continuous-tone still images: Registration of JPEG Profiles, SPIFF Profiles, SPIFF Tags, SPIFF colour Spaces, APPn Markers, SPIFF, Compression types and Registration authorities (REGAUT).

- ITU-T Recommendation T.87 | ISO/IEC 14495-1, Lossless and near-lossless compression of continuous-tone still images-baseline.
- ITU-T Recommendation T.88 | ISO/IEC 14492-1, Lossy/lossless coding of bi-level images

2.2 Additional references

- ISO/IEC 646:1991, ISO 7-bit coded character set for information interchange.
- ISO 5807:1985, Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.
- International Color Consortium, ICC profile format specification. ICC.1:1998–09
- International Electrotechnical Commission. Color management in multimedia systems: Part 2: Colour Management, Part 2–1: Default RGB colour space—sRGB. IEC 61966–2–1 1998. 9 October 1998.
- W3C, Extensible Markup Language (XML 1.0), Rec-xml-19980210
- Digital Imaging Group, Flashpix digital image file format. Version 1.0.1. 10 July 1997

3 Definitions

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

$\lfloor x \rfloor$, floor function, Indicates the largest integer not exceeding x.

$\lceil x \rceil$, ceiling function, Indicates the smallest integer not exceeded by x.

arithmetic coder: An entropy coder used to represent the data compactly.

auxiliary component: A component from the codestream that is used by the application outside the scope of colour space conversion. For example, an opacity component or a depth component would be an auxiliary component.

bit-plane: A two dimensional array of bits. In this Recommendation | Specification a bit-plane refers to all the bits of the same magnitude in all coefficients or samples. Could refer to a bit-plane in a component, tile-component, code-block, region of interest, or other.

bit stream: The actual sequence of bits resulting from the coding of a sequence of symbols. It does not include the marker segments in the main and tile headers.

box: A building block defined by a unique type identifier and length. Some particular boxes may contain other boxes.

box contents: Refers to the data wrapped within the box structure and stored within the DBox field within the Box data structure

box type: Specifies the kind of information that shall be stored with the box and stored within the TBox field within the Box data structure.

cell: An optional subdivision of a tile used for low-memory encoding and decoding.

cleanup pass: A coding pass performed on a single bit-plane of a code-block of coefficients. It is the first pass and only coding pass for the first significant bit-plane; the third and the last pass of all the remaining bit-planes.

codestream: A collection of one or more bit streams and associated (overhead) information required for their decoding and expansion into image data. Such overhead information is restricted to that required for expansion and may include, but is not limited to, marker segments indicating locations of particular bit streams, indicating transform, quantization and coding types, etc.

code-block: A rectangular grouping of coefficients from the same sub-band of a tile-component.

code-block scan: The order in which the coefficients within a code-block are visited during a coding pass. The code-block is processed in stripes, each consisting of four rows and spanning the width of the code-block. Each stripe is processed column by column from left to right and from top to bottom.

coder: An embodiment of either an encoding or decoding process.

coding pass: A complete pass through a code-block where the appropriate coefficient values and context are applied. There are three types of coding passes: significance propagation pass, magnitude refinement pass and cleanup pass.

colour image: An image that has more than one component.

component: An image consists of one or more components. Each component is a two-dimensional array of samples. A colour image typically consists of several components such as RGB.

compressed data: Portion of the codestream, tile, or packet that contains the bit stream description.

conforming reader: An application that reads and interprets a JP2 file correctly as defined by this Recommendation | International Standard.

context: Function of samples previously decoded used to condition the coding of the present sample.

context label: The arbitrary index used to distinguish different context values. The labels are used as a convenience of notation rather than being normative.

context modelling: Procedure determining the probability distribution of difference with predicted bit from the context.

context vector: The binary vector consisting of the significance states of its context neighbor coefficients

decoder: An embodiment of a decoding process, and optionally a colour transformation process.

decoding process: A process which takes as its input compressed data and outputs a reconstructed image data.

decomposition level: A collection of wavelet sub-bands where each coefficient has the same span with respect to the original samples. These include HL, LH, HH and, for the highest decomposition level, LL sub-bands. In this specification, only the LL sub-band can be further decomposed.

discrete wavelet transform: DWT. A wavelet transformation performed on, and resulting in, spatially discrete coefficients.

encoder: An embodiment of an encoding process.

encoding process: A process, which takes as its input a source image and outputs compressed image data.

file format: This consists of a codestream and additional support data and information not explicitly required for the decoding of image data. Examples of such support data are text fields providing titling, security and historical information, markers to support placement of multiple codestreams within a given data file, and markers to support exchange between platforms or conversion to other file formats

HH sub-band: The sub-band obtained by forward horizontal high-pass analysis filtering and vertical high-pass analysis filtering. This sub-band contributes to reconstruction with inverse horizontal low-pass synthesis filtering and vertical low-pass synthesis filtering.

HL sub-band: The sub-band obtained by forward horizontal high-pass analysis filtering and vertical low-pass analysis filtering. This sub-band contributes to reconstruction with inverse horizontal low-pass synthesis filtering and vertical high-pass synthesis filtering.

image: A set of two-dimensional arrays (components) of data.

image area: A rectangular part of the reference grid, registered by offsets from the origin and having the size of the image. The components are contained within this area and are related to the reference grid with respect to this area.

image area offset: The width and height down and to the right of the reference origin where the origin of the image area can be found.

image data: Either source image data or reconstructed image data.

irreversible: A transformation, progression, system, or quantization that, due to systemic or quantization error, disallows lossless recovery signal.

JPEG 2000: Used to refer globally to the encoding and decoding processes in this Recommendation | International Standard and their embodiment in applications.

LH sub-band: The sub-band obtained by forward horizontal low-pass analysis filtering and vertical high-pass analysis filtering. This sub-band contributes to reconstruction with inverse horizontal high-pass synthesis filtering and vertical low-pass synthesis filtering.

LL sub-band: The sub-band obtained by forward horizontal low-pass analysis filtering and vertical low-pass analysis filtering. This sub-band contributes to reconstruction with inverse horizontal high-pass synthesis filtering and vertical high-pass synthesis filtering.

layer: A collection of coding passes from one, or more, code-blocks of a tile-component. Layers have an order for encoding and decoding that must be preserved.

lossless: A descriptive term for the encoding and decoding processes in which the output of the decoding process is identical to the input to the encoding process. Lossless processes require reversible systems.

lossless coding: The mode of operation that refers to any one of the coding processes defined in this Recommendation | International Standard in which all of the procedures are lossless.

lossy: A descriptive term for encoding and decoding processes which are not lossless. This includes both systems that are irreversible and those that include quantization.

magnitude refinement pass: A coding pass performed on a single bit-plane of a code-block of coefficients.

main header: A group of markers and marker segments at the beginning of the codestream that describe the image parameters and universal coding parameters.

marker: A two-byte code in which the first byte is hexadecimal FF (0xFF) and the second byte is a value between 1 (0x01) and hexadecimal FE (0xFE).

marker segment: A marker and associated set of parameters.

mod: $mod(y,x) = z$, where z is an integer such that $0 \le z < x$, and such that $y\text{-}z$ is a multiple of x.

packet: A part of the bit stream comprising a packet header and the coded data from one layer of one decomposition level of one component of a tile.

packet header: Portion of the packet that describes the layer, decomposition level, component, and the code-block segment lengths.

precision: Number of bits allocated to a particular sample.

progressive: The order a codestream where the decoding of each successive bit contributes to a “better” reconstruction of the image. What metrics make the reconstruction “better” is a function of the application. Some examples of progression are increasing resolution or improved pixel fidelity.

quantization: A method of reducing the precision of the individual coefficients to reduce the number of bits used to entropy code them.

reconstructed image (data): An image, which is the output of a decoder.

reconstructed sample (value): The sample value reconstructed by the decoder. This always equals the original sample value in lossless coding but may differ from the original sample value in lossy coding.

reference grid: A regular rectangular array of points to which all other planes (image, component, tile, sub-band, etc.) are associated.

reference tile: A rectangular sub-grid of any size associated with the reference grid.

region of interest: A part of the image, component, or tile-component that is considered more important than the rest.

resolution: The spatial mapping of samples to a physical space. In this Recommendation | Specification the decomposition levels of the wavelet transform relate to each other with different relative resolutions.

reversible: A transformation, progression, system, or quantization that does not suffer systemic or quantization error and, therefore, allows lossless signal recovery.

sample: One element in the two-dimensional array, which comprises a component.

selective arithmetic coding bypass: A coding style where some of the code-block passes are not coded by the arithmetic coder.

shift: Multiplication or division of a binary number by factors of two.

sign-magnitude notation: A binary representation of an integer number where the distance from the origin is expressed with a positive number and the direction from the origin (positive or negative) is expressed with a separate single bit.

significance propagation pass: A coding pass performed on a single bit-plane of a code-block of coefficients.

significance state: Binary state variable that indicates whether the most significant 1 bit of a target coefficient in the current or more significant bit-planes or not.

source image (data): An image used as input to an encoder.

sub-band: A group of transform coefficients resulting from the same sequence of low-pass and high-pass filtering operation, both vertically and horizontally.

sub-band coefficient: A transform coefficient within a given sub-band.

sub-band decomposition: A transformation of an image tile component into sub-bands.

sub-band decomposition level: The number of decompositions performed on the original tile-component samples.

sub-band recomposition: The inverse of sub-band decomposition.

sub-band recomposition level: The remaining number of recompositions needed to reconstruct the original image tile component samples.

superbox: A box that itself contains a contiguous sequence of boxes (and only a contiguous sequence of boxes). As the JP2 file contains only a contiguous sequence of boxes, the JP2 file is itself considered a superbox. When used as part of a relationship between two boxes, the term superbox refers to the box which directly contains the other box.

tile: An array of rectangular points on the reference grid, registered with and offset from the reference grid origin and defined by a base width and height. This tile array overlaps the image area and is used to define image tiles.

tile-component: A rectangular sub-component of any size which is part of a larger component. There is a tile-component for every component and every tile.

tile-part header: A group of markers and marker segments at the beginning of each tile-part codestream that describe the tile-part coding parameters.

tile number: The index of the current tile ranging from zero to the number of tiles minus one.

tile-part: A portion of the codestream that make up some, or all, of a tile. The tile-part includes at least one, and up to all, of the packets that make up the tile.

tile-part number: The tile number of the tile with which the tile-part is associated.

transform: A mathematical mapping from one signal space to another.

transform coefficient: A value that is the result of a transformation.

wavelet transform: A transformation which iteratively transforms one signal into two or more filtered and decimated signals corresponding to different frequency bands.

XOR: Exclusive OR logical operator.

4 Abbreviations

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

CCITT: International Telegraph and Telephone Consultative Committee, now ITU-T

ICC: International Colour Consortium

IEC: International Electrotechnical Commission

ISO: International Organization for Standardization

ITTF: Information Technology Task Force

ITU: International Telecommunication Union

ITU-T: International Telecommunication Union – Telecommunication Standardization Sector (formerly the CCITT)

JPEG: Joint Photographic Experts Group **-** The joint ISO/ITU committee responsible for developing standards for continuous-tone still picture coding. It also refers to the standards produced by this committee: ITU-T T.81 | ISO/IEC 10918-1, ITU-T T.83 | ISO/IEC 10918-2, ITU-T T.84 | ISO/IEC 10918-3 and T.87 | ISO/IEC 14495.

JURA: JPEG Utilities Registration Authority

1D-DWT: One dimensional discrete wavelet transform

DWT: Discrete wavelet transform

FDWT: Forward discrete wavelet transform

IDWT: Inverse of the forward discrete wavelet transform

LSB: Least significant bit.

MSB: Most significant bit.

PCS: Profile Connection Space

SNR: Signal to noise ratio. In this context it usually refers to reconstructed sample fidelity.

5 Symbols

For the purposes of this Recommendation | International Standard, the following symbols apply.

0x-: Denotes a hexadecimal number.

CME: Comment and extension marker

COC: Coding style component marker

COD: Coding style default maker

EOI: End of image marker

IEM: Packet, main marker

IET: Packet, tile marker

QCC: Quantization component marker

QCD: Quantization default marker

RIM: Region of interest marker

SIZ: Size of image marker

SOC: Start of image (codestream) marker

SOS: Start of scan marker

SOT: Start of tile marker

TLM: Tile length marker

URI: Uniform Resource Identifier

URL: Uniform Resource Location

6 General description

This specification describes an image compression system that allows great flexibility, not only for the compression of images, but also for the access into the compressed data. The codestream provides a number of mechanisms for locating and extracting data for the purpose of retransmission, storage, display, or editing. This access allows the storage and the retrieval of data appropriate for a given application, without decoding.

The division of the both the original data and the compressed data in a number of ways leads to the ability to extract data from the compressed codestream to form reconstructed images with lower resolution, lower bit-rate, or regions of the original images. This allows the matching of a codestream to the transmission channel, storage device, or display device, regardless of the size, number of components, and sample precision of the original image. The codestream can be manipulated without decoding to achieve a more efficient arrangement for a given application.

Thus, the sophisticated features of this specification allow a single codestream to be used efficiently by a number of applications. The largest image source devices can provide a codestream that is easily processed for the smallest image display device, for example.

6.1 Purpose

There are three main elements described in this Recommendation | International Standard:

Encoder: An embodiment of an encoding process. An encoder takes as input digital source image data and parameter specifications, and by means of a set of procedures generates as output compressed image data.

Decoder: An embodiment of a decoding process and a sample transformation process. A decoder takes as input compressed image data and parameter specifications, and by means of a specified set of procedures generates as output digital reconstructed image data.

Codestream format: A compressed image data representation that includes all parameter specifications used in the encoding process. The interchange format is for exchange between application environments. The interchange format is specified in Annex. A

6.2 Coding principles

The main procedures for this Recommendation | International Standard are shown in Figure 6-1. Procedures are presented in the Annexes in the order of the decoding process.

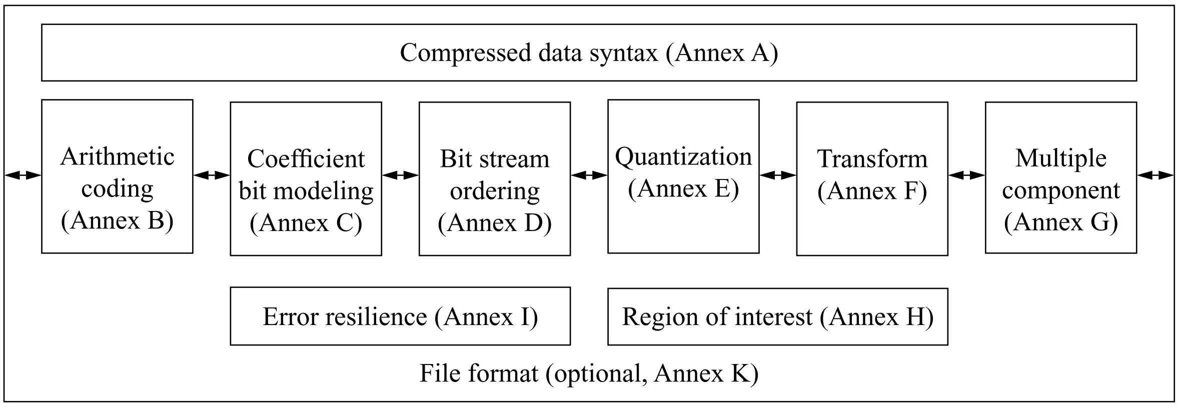

Compressed data syntax (Annex A)
Arithmetic coding (Annex B)
Coefficient bit modeling (Annex C)
Bit stream ordering (Annex D)
Quantization (Annex E)
Transform (Annex F)
Multiple component (Annex G)
Error resilience (Annex I)
Region of interest (Annex H)
File format (optional, Annex K)

Figure 6-1 — Specification block diagram

The key to this specification is the multiple ways that both the image and the codestream can be divided. Many images have multiple components. This specification has a facility to decorrelate three component planes. Other than that, the components can be extracted and reconstructed without regard to the other components. (See Annex G.)

The image may be divided into tiles. These tiles are rectangular arrays that include the same relative portion of all the components. These tile-components can also be extracted and reconstructed independently. This tile independence provides one of the methods for extracting a region of the image. (See Annex D.)

The tile-components are decomposed into different decomposition levels using a wavelet transform. These decomposition levels contain a number of sub-bands populated with coefficients that describe the horizontal and vertical spatial frequency characteristics of the original tile-component planes. The coefficients provide local frequency information. That is, a small number of coefficients completely describe a single sample. A decomposition level is related to the next decomposition level by spatial powers of two. That is, each successive decomposition level of the sub-bands is has approximately half the horizontal and half the vertical resolution of the previous. Images of lower resolution than the original are generated by decoding a selected sub-set of these sub-bands. (See Annex F.)

Although there are the same number of coefficients as samples, the information content tends to be concentrated in just a few coefficients. Through quantization, the information content of a large number of small magnitude coefficients is further reduced. Additional processing reduces the bits required to represent these quantized coefficients, sometimes significantly compared to the original image. With the same bits, however, the reconstructed image suffers proportionally less loss in terms of visual quality. This holds true for a large enough range of bit-rates and quality factors to make this image compression system useful for a number of applications. (See Annex E.)

The individual sub-bands of a tile-component are further divided into code-blocks. These rectangular arrays of coefficients can be extracted independently. The individual bit-planes of the coefficients in a code-block are coded with three coding passes. Each of these coding passes collects contextual information about the bit-plane data. (See Annex C.) An arithmetic coder uses this contextual information, and its internal state, to decode a compressed bit-stream. (See Annex B.) Different termination mechanisms allow different levels of independent extraction of this coding pass data.

The bit stream data created from these coding passes is conceptually grouped in layers. Layers are an arbitrary number of groupings of coding passes from each code-block. Although there is great flexibility in layering, the basic premise is that each successive layer contributes to a higher quality image. (See Annex D.)

Packets are a fundamental unit of the compressed codestream. A packet is a particular partition of one layer of one decomposition level of one tile-component. This partition provides another method for extracting a spatial region independently from the codestream. These packets may be interleaved in the codestream using a few different methods. (See Annex D.)

A mechanism is provided that allows the data corresponding to regions of interest in the original tile-components to be coded and placed earlier in the bit stream. (See Annex H.)

Several mechanisms are provided to allow the detection and concealment of bit errors that might occur over a noisy transmission channel. (See Annex I.)

The compressed data relating to a tile, organized in packets, are arranged in one, or more, tile-parts. A tile-part header, comprising a series of markers and marker segments, contains information about the various mechanisms and coding styles that are needed to locate, extract, decode, and reconstruct every tile-component. At the beginning of the entire codestream is a main header, comprised of markers and marker segments, that offer similar information as well as information about the original image. (See Annex A.)

The codestream is wrapped in a file format that allows applications to interpret the meaning of, and other information about, the image. The file format may also contain other data besides the codestream. (See Annex J.)

To review, procedures that divide the original image are the following:

- The image is decomposed into components.
- The image and its components are decomposed into rectangular tiles. The tile-component is the basic unit of the original or reconstructed image.
- Performing the wavelet transform on a tile-component creates decomposition levels that are related to the original resolution by powers of two.
- These decomposition levels are made up of sub-bands of coefficients that describe the local frequency characteristics of the tile-component.
- The sub-bands of coefficients are quantized and collected into rectangular arrays of code-blocks.
- The bit-planes of the coefficients in a code-block are entropy coded in three coding passes.
- Some of the coefficients can be coded first to provide a region of interest.

At this point the data is fully decomposed and coded. The procedures that reassemble these bit stream units into the codestream are the following:

- The coding-passes from the code-blocks are collected in layers.
- Packets are composed of a single partition of a single layer of a single decomposition level of a single tile-component. The packets are the basic unit of the compressed data.
- All the packets from a tile are interleaved in one of several orders and placed in one, or more, tile-parts.
- The tile-parts have a descriptive tile-part header and can be interleaved in any order.
- The codestream has a main header at the beginning that describes the original image and the various recomposition and coding styles that shall be used to locate, extract, decode, and reconstruct the image that is appropriate.
- The file format describes the meaning of the image and its components in the context of the application.

7 Interchange format requirements

The interchange format is the coded representation of compressed image data for exchange between application environments. This specification differentiates between file format and codestream syntax (Annex A). The codestream syntax is the required description of the coded data in the codestream. The file format contains medatata about the image in addition to the codestream, for example to allow for visualization, printing.

The interchange format requirements are that any compressed image data represented in interchange format shall comply with the syntax and code assignments appropriate for the coding processes defined in this Recommendation | International Standard, as specified in Annex A.

8 Encoder requirements

An encoding process converts source image data to compressed image data. Annexes A, B, C, D, E, F, G, and H describe the encoding process. Note that all encoding processes are specified informatively.

An encoder is an embodiment of the encoding process. In order to conform to this Recommendation | International Standard, an encoder shall convert source image data to compressed image data, which conform to the interchange format syntax specified in Annex A.

9 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Annexes A, B, C, D, E, F, G, and H describe and specify the decoding process.

A decoder is an embodiment of the decoding process. In order to conform to this Recommendation | International Standard, a decoder shall convert all, or specific parts of any compressed image data which conforms to the interchange format syntax specified in Annex A to a reconstructed image that is related to the source image.

Annex A

Compressed data syntax

(This annex forms an integral part of this Recommendation | International Standard)

This Annex specifies the marker and marker segment syntax defined by this Recommendation | International Standard. These marker segments provide codestream signaling for this Recommendation | International Standard. Further, it anticipates signaling needed for future specifications that include this Recommendation | International Standard as a normative reference.

A.1 Headers and marker segments

This Recommendation | International Standard uses marker segments to delimit and signal the characteristics of the codestream. This set of markers and marker segments is the minimal information needed to achieve the features of this Recommendation | International Standard and is not a complete or generic file format.

A.1.1 Markers and marker segments

Every marker is two bytes long. The first byte is all ones or 0xFF. The second byte denotes which marker and ranges between 0x01 and 0xFE. Many of these markers are already used in ITU-T Rec. T.81 | ISO/IEC 10918-1 and ITU-T Rec. T.84 | ISO/IEC 10918-3.

A marker segment includes a marker and associated parameters. Many marker segments just contain the marker. If there are more parameters in a marker segment than just the marker then there is a two byte value that denotes the length of the marker segment (not including the marker).

A.1.2 Types of marker segments

Six types of marker segments are used: delimiting, fixed information, functional, in bit stream, pointer, and informational marker segments. Delimiting marker and marker segments must be used to frame the headers and the data. Markers are only used for delimiting. However, there are delimiting marker segments as well.

Fixed information marker segments give required information about an image. The location of these marker segments, like delimiting marker segments, is specified.

Functional marker segments must be used to describe the coding functions used.

In bit stream markers are used for error resilience. These marker segments are the only ones without a length field.

Pointer marker segments point to specific places in the bit stream.

Informational marker segments provide optional information about the data.

A.1.3 Syntax compatibility with ITU.T Rec.T.81 | IS0/IEC 10918-1

The marker segment syntax uses the same construction as the JPEG standard. Some of the marker segments are exactly the same. Those that are not, have numbers that were reserved in ITU.T Rec. 81 | IS 10918-1 and are registered by

JURA. Some of this marker field (0xFF30 — 0xFF3F) is reserved by this specification as markers without marker segments. This will enable backward compatibility. Table A-1 shows in which specification these marker are defined.

Table A-1 — Marker definitions

Marker value range	Standard definition
0xFF00, 0xFF01, 0xFFFE, 0xFFC0 — 0xFFDF	Defined in ITU-T Rec. T.81 | ISO/IEC 10918-1
0xFFF0 — 0xFFF6	Defined in ITU-T Rec. T.84 | ISO/IEC 10918-3
0xFFF7 — 0xFFF8	Defined in ITU-T Rec. T.87 | ISO/IEC 14495-1
0xFF4F — 0xFF6F	Defined in this International Standard | Recommendation
0xFF30 — 0xFF3F	Reserved for definition as markers only (no marker segments)

A.1.4 Marker segment and codestream rules

- Marker segments, and therefore the headers, are a multiple of 8 bits (one byte). Further, the bit stream data between the headers are padded to also be aligned to a multiple of 8 bits.
- All marker segments in a tile-part header apply only to the tile to which it belongs.
- All marker segments in the main header apply to the whole image unless specifically overridden by marker segments in a tile-part header.
- Packets are located at 8 bit (one byte) boundaries. The data before a packet are padded with enough extra bits to make a multiple of 8 bits.
- Delimiting and fixed information marker segments must appear at specific points in the codestream.
- The marker segments should always correctly describe the image as represented by the codestream. If truncation, alteration, or editing of the codestream has been performed, the marker segments shall be updated accordingly.
- Marker segments include a length parameter (except for some delimiting markers) that indicates the length of marker segment, excluding the two byte marker. This allows a decoder to skip over a marker and marker segment as needed.
- All parameter values in marker segments are big endian (most significant byte first).
- All markers with the marker value between 0xFF30 to 0xFF3F are defined to only have the two byte marker, i.e. there are no length parameter or other parameters.

A.1.5 Key to graphical descriptions

Each marker segment is described in terms of function, usage, and length. The function describes the information contained in the marker segment. The usage describes the logical location and frequency of this marker segment in the codestream. The length describes which parameters determine the length of the marker segment.

These descriptions are followed by a figure that shows the order and relationship of the parameters in the marker segment. Figure A-1 shows an example of this type of figure. The marker segments are designated by a three letter abbreviation. The parameter values have capital letter designations with the markers abbreviation as a sub-script. A rectangle is used to indicate the parameters in the marker segment. The width of the rectangle is proportional to the number of bytes in the field. A shaded rectangle indicates that the parameter is of varying size. Two parameters with superscripts and a gray area between indicate a run of several of these parameters. A dashed rectangle indicates that the parameter is only present if necessary.

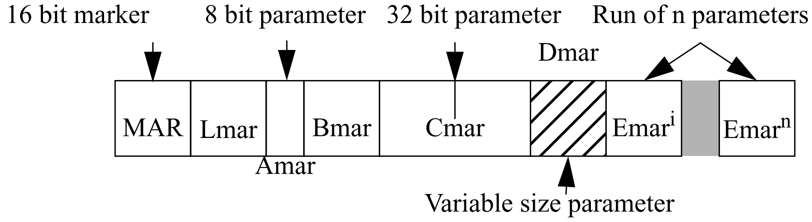


Figure A-1 — Example of the marker segment description figures

The meaning of each parameter in the marker segment is defined in a list. If there is a run of the same parameter, the length and nature of that run is defined. The first rectangle is the marker value and is always 16 bits. The second rectangle is the length and is 16 bits.

There is a table that describes the parameter values or references which refer to other tables. Tables for individual parameters are provided to describe any parameter without a simple numerical value. The term “use Table A-x” will be used to specify a parameter. The term “refer to Table A-x” means that Table A-x will define which other table describes a parameter.

Some marker segments use an Sxxx and SPxxx syntax for the XXX marker segment. The Sxxx parameter selects between many possible states. According to this selection, the SPxxx parameter, or parameter, list is modified.

A.2 Information in the marker segments

Table A-2 lists the markers specified in this Recommendation | International Standard. Table A-3 shows a list of the information provided by the syntax and which marker segment contains that information.

Table A-2 — List of marker segments

	Name	Code	Main header	Tile-part header
Delimiting marker segments				
Start of codestream	SOC	0xFF4F	required	x[a]
Start of tile-part	SOT	0xFF90	x	required
Start of data	SOS	0xFFDA	x	last marker
End of image[b]	EOI	0xFFD9		x
Fixed information marker segments				
Image and tile size	SIZ	0xFF51	required	x
Functional marker segments				
Coding style default	COD	0xFF52	required	optional
Coding style component	COC	0xFF53	optional	optional
Region-of-interest	RGN	0xFF5E	optional	optional

Table A-2 — List of marker segments

	Name	Code	Main header	Tile-part header
Quantization default	QCD	0xFF5C	required	optional
Quantization component	QCC	0xFF5D	optional	optional
Progression order change, main	POM	0xFF5D	optional[c]	x
Progression order change, tile	POT	0xFF5F	x	optional[c]
Error resilience style	ERS	0xFF59	optional	optional
Pointer marker segments				
Tile-part lengths, main header	TLM	0xFF55	optional	x
Packet length, main header	PLM	0xFF57	optional	x
Packet length, tile-part header	PLT	0xFF58	x	optional
Packed packet headers, main header	PPM	0xFF60	optional[d]	x
Packed packet headers, tile-part header	PPT	0xFF61	x	optional[d]
In bit stream marker segments				
Start of partition	SOP	0xFF91		optional, in bit stream
Informational marker segments				
Comment and extension	CME	0xFF64	optional	optional

a. Required means the marker segment shall be in this header, optional means it may be, and “x” means not used in this header.

b. The EOI marker is the last in the image. It is in neither the main nor the tile-part headers.

c. Either the POM or POT marker segment is required if there are progression order changes. If the POM marker segment is used then POT marker segments shall not be used, and visa versa.

d. Either the PPM or PPT marker segment is required if there packet headers are not distribuited in the bit stream. If the PPM marker segment is used then PPT marker segments shall not be used, and visa versa.

Table A-3 — Information in the marker segments

Information	Marker segment
Capabilities Image size or reference grid size (height and width) Tile size (height and width) Number of components Decorrelating transform used Component precision Component mapping to the reference grid (sub-sampling)	SIZ
Tile number Tile-part data length	SOT, TLM

Table A-3 — Information in the marker segments

Information	Marker segment
Coding style (wavelet coding, uncoded, etc.) Number of decomposition levels Progression style Number of layers Code-block size Coding contexts Wavelet transform	COD, COC
Region of interest location Region of interest size Region of interest shift	RGN
No quantization Quantization implicit Quantization explicit	QCD, QCC
Progression point (layer, resolution, component) Progression order change	POM, POT
Error resilience	ERS, SOP
Code-block values for new layers Code-block layer number Code-block inclusion Maximum bit depth Truncation point Bit stream length for decomposition level and layer in a block	packet header, PPM, PPT
Packet lengths	PLM, PLT
Optional information	CME

A.3 Construction of the codestream

Figure A-2 shows the construction of the codestream. Figure A-3 shows the main header construction. Note that all of the solid lines show required marker segments. The marker segments on the left are required to be in a specific location. The dashed lines show optional or possibly not required marker segments. Figure A-4 shows the construction of a tile-part header.

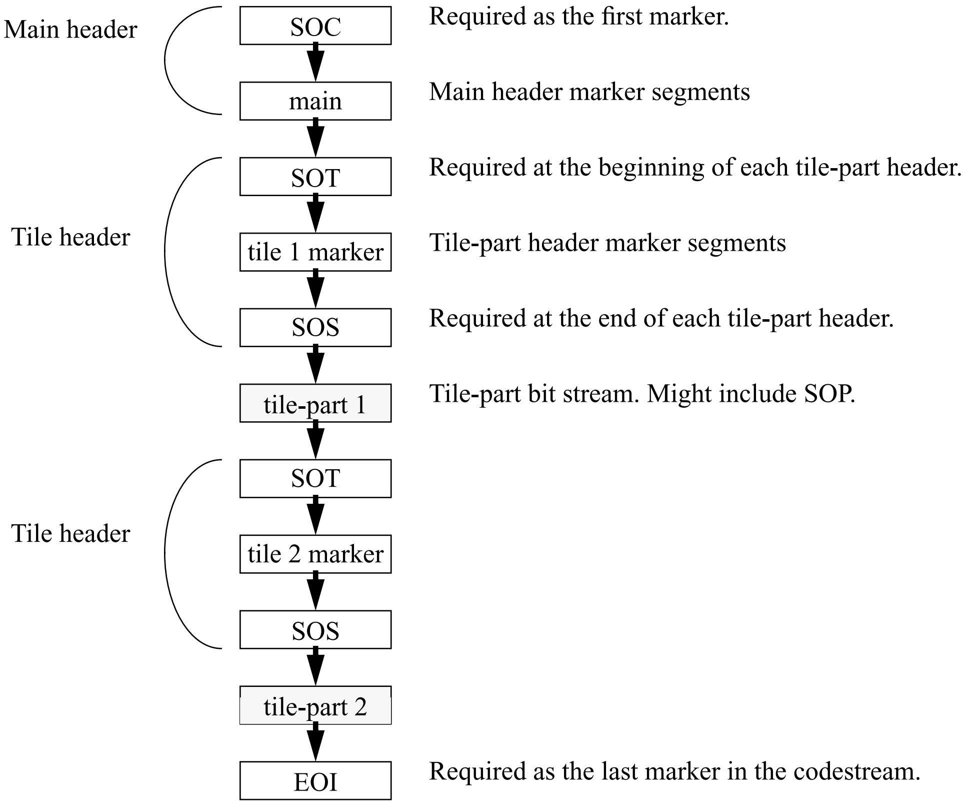
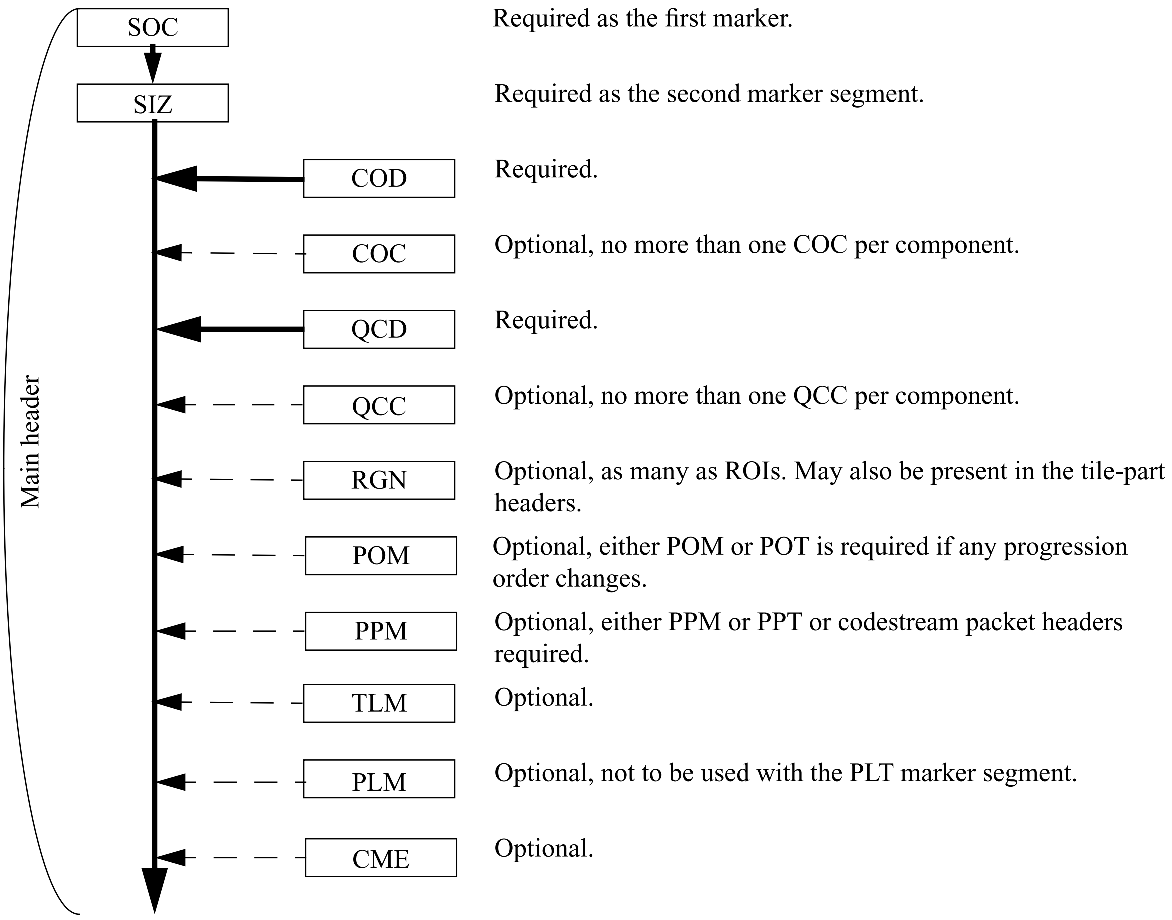
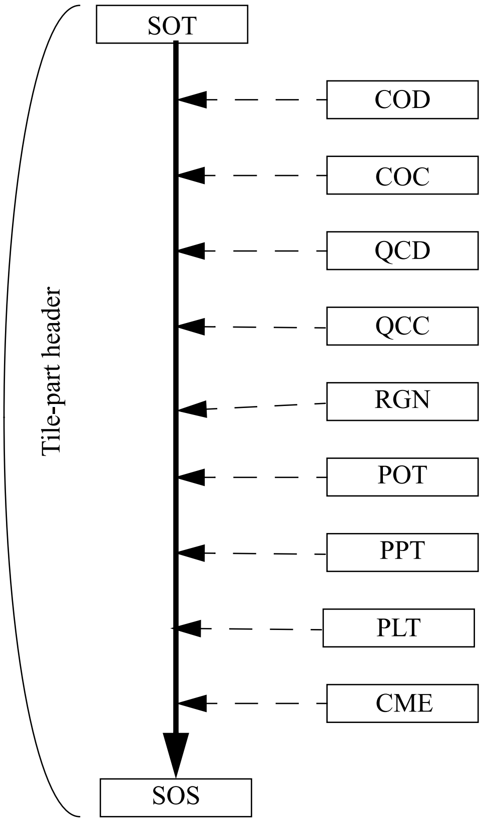

Main header
SOC
Required as the first marker.
main
Main header marker segments
SOT
Required at the beginning of each tile-part header.
Tile header
tile 1 marker
Tile-part header marker segments
SOS
Required at the end of each tile-part header.
tile-part 1
Tile-part bit stream. Might include SOP.
SOT
Tile header
tile 2 marker
SOS
tile-part 2
EOI
Required as the last marker in the codestream.

Figure A-2 — Construction of the codestream



SOC	Required as the first marker.
SIZ	Required as the second marker segment.
COD	Required.
COC	Optional, no more than one COC per component.
QCD	Required.
QCC	Optional, no more than one QCC per component.
RGN	Optional, as many as ROIs. May also be present in the tile-part headers.
POM	Optional, either POM or POT is required if any progression order changes.
PPM	Optional, either PPM or PPT or codestream packet headers required.
TLM	Optional.
PLM	Optional, not to be used with the PLT marker segment.
CME	Optional.

Figure A-3 — Construction of the main header



Required as the first marker segment of every tile-part header.

Optional, no more than one per tile-part.

Optional, no more one COC per component.

Optional, no more than one per tile-part.

Optional, no more one QCC per component.

Optional, as many as ROIs in tile-part. May also be present in the main header.

Optional, either POM or POT required if any progression order changes.

Optional, either PPM or PPT or codestream packet headers required.

Optional, not to be used with the PLM marker segment.

Optional.

Required as the last marker of every tile or tile-part header.

Figure A-4 — Construction of a tile-part header

The COD and COC marker segments and the QCD and QCC marker segments have hierarchy of usage. This is designed to allow tile-components to have dissimilar coding and quantization characteristics with a minimum of signalling.

For example, the COD marker segment is required in the main header. If all components in all the tiles are coded the same way, this is all that is required. If there is one component that is coded differently than the others (for example the luminance component of an image composed of luminance and chrominace components) then the COC can denote that in the main header. If one or more components are coded differently in different tiles, then the COD and COC are used in a similar manner to denote this in the tile header.

A.4 Delimiting markers

The delimiting marker segments are strictly required. Each codestream has only one SOC marker, one EOI marker, and at least one tile-part (SOT and SOS). Each tile-part has one SOT and one SOS marker. The SOC, SOS, and EOI delimiting markers are 16 bits in length with no explicit length information.

A.4.1 Start of codestream (SOC)

Function: Marks the beginning of a codestream specified in this International Standard | Recommendation.

Usage: Shall be the first marker in codestream. There shall be only one SOC per codestream.

Length: Fixed.

SOC: Marker value.

Table A-4 — Start of codestream parameter values

Parameter	Size (bits)	Values
SOC	16	0xFF4F

A.4.2 Start of tile-part (SOT)

Function: Marks the beginning of a tile-part and the index of that tile within a codestream.

Usage: Shall be the first marker segment in a tile-part section. There shall be at least one SOT in a codestream. There shall be only one SOT per tile-part.

Length: Fixed.

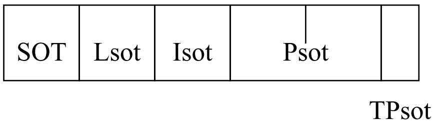


Figure A-5 — Start of tile-part syntax

SOT: Marker value. Table A-5 shows the size and values for start of tile-part.

Lsot: Length of marker segment in bytes (not including the marker).

Isot: Tile number. This number refers to the tiles in raster order starting at the number 0.

Psot: Length, in bytes, from the beginning of the first byte of this SOT marker segment of the tile-part to the end of the data of that tile-part. Figure A-15 shows this alignment.

TPsot: Tile-part instance. If this is a tile-part there is a specific order required for decoding tile-parts, this index denotes the order from 0. If there is only one tile-part for a tile then this value is zero.

Table A-5 — Start of tile-part parameter values

Parameter	Size (bits)	Values
SOT	16	0xFF90
Lsot	16	9
Isot	16	0 — 65534
Psot	32	0 — (2^{32}-2)
TPsot	8	0 — 255

A.4.3 Start of data (SOS)

Function: Indicates the beginning of bit stream data for the current tile-part. The SOS also indicates the end of a tile-part header.

NOTE — This marker shares the same number as the SOS marker in ITU-T Rec. T.81 | ISO/IEC 10918-1.

Usage: Shall be the last marker in a tile-part header. Data between an SOS and the next SOT or EOI (end of image) shall be a multiple of 8 bits — the codestream is padded with bits as needed (see Annex C.5.2). There shall be at least one SOS in a codestream. There shall be one SOS per tile-part.

Length: Fixed.

SOS: Marker value

Table A-6 — Start of data parameter values

Parameter	Size (bits)	Values
SOS	16	0xFFDA

A.4.4 End of image (EOI)

Function: Indicates the end of the image. EOI acts as a check to ensure that the stream is still byte aligned.

NOTE — This marker shares the same number as the EOI marker in ITU-T Rec. T.81 | ISO/IEC 10918-1.

Usage: Shall be the last marker in a codestream. There shall be one EOI per codestream.

Length: Fixed.

EOI: Marker value

Table A-7 — End of image parameter values

Parameter	Size (bits)	Values
EOI	16	0xFFD9

A.5 Fixed information marker segment

This marker segment describes required information about the image.The SIZ marker segment is required in the main header immediately after the SOC marker segment.

A.5.1 Image and tile size (SIZ)

Function: Provides information about the uncompressed image such as the width and height of the reference grid, the width and height of the tiles, the number of components, component size (depth), and how the components span the reference grid.

Usage: Shall be one and only one in the main header immediately after the SOC marker segment. There shall be only one SIZ per codestream.

Length: Variable depending on the number of components.

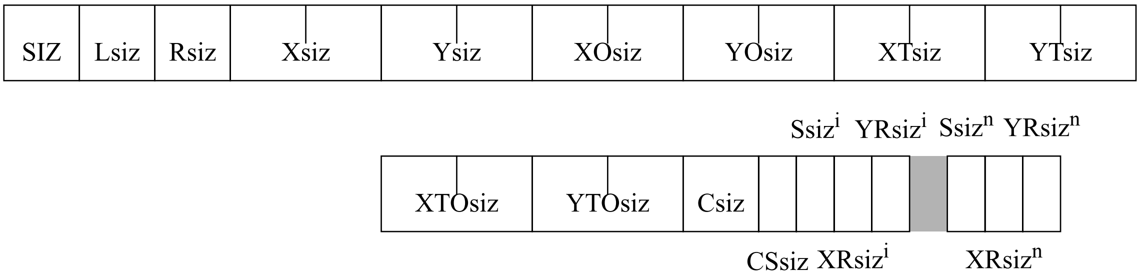


Figure A-6 — Image and tile size syntax

SIZ: Marker value. Table A-8 shows the size and parameter values for image and tile size.

Lsiz: Length of marker segment in bytes (not including the marker).

Rsiz: Denotes capabilities of the codestream.

Xsiz: Width of the reference grid. Optimally, this value provided the least common multiple for all the XRsiz parameters.

Ysiz: Height of the reference grid. Optimally, this value provided the least common multiple for all the YRsiz parameters.

XOsiz: Horizontal offset from the origin of the reference grid to the left side of the image area.

YOsiz: Vertical offset from the origin of the reference grid to the top side of the image area.

XTsiz: Width of one reference tile with respect to the reference grid.

YTsiz: Height of one reference tile with respect to the reference grid.

XTOsiz: Horizontal offset from the origin of the reference grid to the left side of the first tile.

YTOsiz: Vertical offset from the origin of the reference grid to the top side of the first tile.

Csiz: Number of components in the image.

CSsiz: Multiple component transformation used.

Ssiz^i: Precision (depth) in bits of the ith component. (If a multi-component transform is performed, this is the precision afterwards.) This parameter is repeated for all components. The most significant bit of this field indicates whether the component is signed or not, a one bit indicates signed components. The remaining seven bits indicate the component depth which must be between 1 and 127 inclusive. One value for each component.

XRsiz^i: Horizontal separation of a sample of ith component with respect to the reference grid. One value for each component.

YRsiz^i: Vertical separation of a sample of ith component with respect to the reference grid. One value for each component.

Table A-8 — Image and tile size parameter values

Parameter	Size (bits)	Values
SIZ	16	0xFF51
Lsiz	16	42 — 65534
Rsiz	16	use Table A-9
Xsiz	32	1 — (2^{32} - 1)
Ysiz	32	1 — (2^{32} - 1)
XOsiz	32	1 — (2^{32} - 2)
YOsiz	32	1 — (2^{32} - 2)
XTsiz	32	1 — (2^{32} - 1)
YTsiz	32	1 — (2^{32} - 1)
XTOsiz	32	0 — (2^{32} - 2)
YTOsiz	32	0 — (2^{32} - 2)
Csiz	16	1 — 16384
CSsiz	8	use Table A-10
Ssiz^i	8	use Table A-11
XRsiz^i	8	1 — 255
YRsiz^i	8	1 — 255

Table A-9 — Capability Rsiz parameter

Value (bits) MSB LSB	Capability
0000 0000 0000 0000	Capabilities specified in this Recommendation | International Standard only
	All other values reserved

Table A-10 — Multiple component transformation CSsiz parameter

Values (bits) MSB LSB	Multiple component transformation type
0000 0000	No multiple component transform specified. (A multiple component transform may be specified by the file format level.)

Table A-10 — Multiple component transformation CSsiz parameter

Values (bits) MSB LSB	Multiple component transformation type
0000 0001	Reversible component transform on components 0, 1, 2 (see Annex G.1)
0000 0010	YCbCr transform on components 0, 1, 2 (see Annex G.1)
	All other values reserved

Table A-11 — Component Ssiz parameter

Values (bits) MSB LSB	Coefficient size
x000 0001 x111 1111	Components are 1 bit deep through 127 bits deep respectively
0xxx xxxx	Components are unsigned values
1xxx xxxx	Components are signed values
	All other values reserved.

A.6 Functional marker segments

These marker segments describe the functions used to code the entire tile, if found in the tile-part header, or image, if found in the main header. If there are multiple tile-parts in a tile, then these marker segments shall be found only in the first tile-part.

A.6.1 Coding style default (COD)

Function: Describes the coding style, decomposition, and layering used for compressing all components of an image (if in the main header) or a tile (if in the tile-part header). The parameter values can be overridden for an individual component by a COC marker segment in either the main or tile-part header.

Usage: Shall be one and only one in the main header. May be one and only one in each tile-part headers. If there are multiple tile-parts in a tile, and this marker segment is present, it shall be found only in the first tile-part.

When used in the main header, the COD marker segment parameter values are used for all tile-components that do not have a corresponding COC marker segment in either the main or tile-part header. When used in the tile-part header it overrides the main header COD and COCs and is used for all components in that tile without a corresponding COC marker segment in the tile-part. Thus, the hierarchy is the following:

Tile-part COC > Tile-part COD > Main COC > Main COD

Length: Variable depending on the value of Scod.

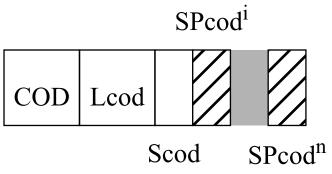


Figure A-7 — Coding style default syntax

COD: Marker value. Table A-12 shows the size and parameter values for coding styles.

Lcod: Length of marker segment in bytes (not including the marker).

Scod: Coding style for all components. Table A-13 shows the value for the Scod parameter.

SPcod^i: Parameters for coding style designated in Scod. The parameters are designated, in order from top to bottom, in the appropriate table.

Table A-12 — Coding style default parameter values

Parameter	Size (bits)	Values
COD	16	0xFF52
Lcod	16	4 — 65534
Scod	8	use Table A-13
SPcod^i	variable	refer to Table A-13

Table A-13 — Coding style parameter values for the Scod parameters

Values (bits) MSB LSB	Coding style	SPcod usage
0000 0000	Entropy coder, PARTITION = 0	use Table A-14
0000 0001	Entropy coder, PARTITION = 1	use Table A-14
	All other values reserved	

Table A-14 — Coding style parameter values from SPcod parameters

Parameters (in order)	Size (bits)	Values	Meaning of SPcod values
Decomposition levels	8 bits	0 — 255	Number of decomposition levels, dyadic decomposition. (Zero implies no transform.)
Progression style	8 bits	use Table A-15	Progression style
Number of layers	16 bits	0 — 65535	Number of layers
Code-block size width	8 bits	use Table A-16	Code-block width exponent value, xcb
Code-block size height	8 bits	use Table A-16	Code-block height exponent value, ycb
Code-block context	8 bits	Table A-17	Style of the code-block coding passes
Transform	8 bits	use Table A-18	Wavelet transform used.
Packet partition size	0 bits variable	use Table A-19	If PARTITION = 0, no value If PARTITION = 1, indicates partition size width and height, repeat for every decomposition level

Table A-15 — Progression style for the SPcod, Ppom, and Ppot parameters

Values (bits) MSB LSB	Progression style
0000 0000	Layer progressive
0000 0001	Resolution-layer progressive
0000 0010	Resolution-position progressive
0000 0011	Position-component progressive
0000 0100	Component-position progressive
	All other values reserved

Table A-16 — Width and height of the code-blocks for the SPcod and SPcoc parameters

Values (bits) MSB LSB	Code-block width or height
0000 0000 — 0000 1000	Offset exponent value for code-block width or height (xcb=value+2 or ycb=value+2)
	All other values reserved

Table A-17 — Code-block style for the SPcod and Scoc parameters

Values (bits) MSB LSB	Code-block style
xxxx xxx0 xxxx xxx1	No selective arithmetic coding bypass Selective arithmetic coding bypass
xxxx xx0x xxxx xx1x	No reset of context probabilities on coding pass boundaries Reset context probabilities on coding pass boundaries
xxxx x0xx xxxx x1xx	No termination on each coding pass Termination on each coding pass
xxxx 0xxx xxxx 1xxx	No vertically stripe causal context Vertically stripe causal context
xxx0 xxxx xxx1 xxxx	No predictable termination Predictable termination
	All other values reserved

Table A-18 — Transform for the SPcod and Scoc parameters

Values (bits) MSB LSB	Transform type
0000 0000	9-7 irreversible wavelet transform
0000 0001	5-3 reversible wavelet transform
	All other values reserved

Table A-19 — Packet partition width and height for the SPcod parameters

Values (bits) MSB LSB	Packet partition size
xxxx 0000 — xxxx 1111	Packet partition width exponent, PPx
0000 xxxx — 1111 xxxx	Packet partition height exponent, PPy

A.6.2 Coding style component (COC)

Function: Describes the coding style, decomposition, and layering used for compressing a particular component.

Usage: Optional in both the main and tile-part headers. No more than one per component may be present in either the main or tile-part headers. If there are multiple tile-parts in a tile, and this marker segment is present, it shall be found only in the first tile-part.

When used in the main header it overrides the main COD marker segment for the specific component. When used in the tile-part header it overrides the main COD, main COC, and tile COD for the specific component. Thus, the hierarchy is the following:

Tile-part COC > Tile-part COD > Main COC > Main COD

Length: Variable depending on the value of Scoc.

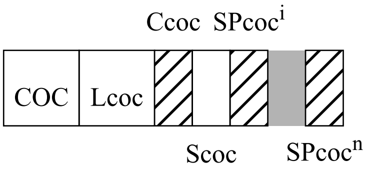


Figure A-8 — Coding style component syntax

COC: Marker value. Table A-20 shows the size and parameter values for coding styles.

Lcoc: Length of marker segment in bytes (not including the marker).

Ccoc: The number of the component to which this marker segment relates. The components are numbered 0, 1, 2, etc. (Either 8 or 16 bits depending on Csiz value.)

Scoc: Coding style for this component. Table A-21 shows the value for each Scoc parameter.

SPcoc^{i}: Parameters for coding style designated in Scoc. The parameters are designated, in order from top to bottom.

Table A-20 — Coding style component parameter values

Parameter	Size (bits)	Values
COC	16	0xFF53
Lcoc	16	5 — 65534
Ccoc	8 16	0 — 255; if Csiz < 257 0 — 65534; Csiz ≥ 257
Scoc	8	use Table A-21
SPcoc^{i}	variable	refer to Table A-21

Table A-21 — Coding style parameter values for the Scoc parameters

Values (bits) MSB LSB	Coding style	SPcoc usage
0000 0000	Entropy coder, PARTITION = 0	use Table A-22

Table A-21 — Coding style parameter values for the Scoc parameters

Values (bits) MSB LSB	Coding style	SPcoc usage
0000 0001	Entropy coder, PARTITION = 1	use Table A-22
	All other values reserved	

Table A-22 — Coding style parameter values from SPcoc parameters

Parameters (in order)	Size (bits)	Values	Meaning of SPcoc values
Decomposition levels	8 bits	0 — 255	Number of decomposition levels, dyadic decomposition. (Zero implies no transform.)
Code-block size width	8 bits	use Table A-16	Code-block width exponent value, xcb
Code-block size height	8 bits	use Table A-16	Code-block height exponent value, ycb
Code-block context	8 bits	Table A-17	Style of the code-block coding passes
Transform	8 bits	use Table A-18	Wavelet transform used.
Packet partition size	0 bits variable	use Table A-19	If PARTITION = 0, no value If PARTITION = 1, indicates partition size width and height, repeat for every decomposition level

A.6.3 Region-of-interest (RGN)

Function: Signals the location, shift, and type of RGN in the codestream.

Usage: May be used in the main or tile-part header. If used in the main header it refers to an ROI of the whole image, regardless of tiling. In other words, the RGN defined in the main header could imply ROIs in several tiles. In fact, entire tiles could be part of the ROI. When used in the tile-part header only that tile is affected. Redundant description of a single RGN (defined in both the main and tile-part headers) is not permitted. There can be multiple uses of this marker segment in both the main and tile-part headers. Each use implies a separate RGN. It there are multiple tile-parts in a tile, then this marker segment shall be found only in the first tile-part.

Length: Variable.

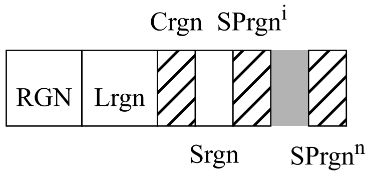


Figure A-9 — Coding style default syntax

RGN: Marker value. Table A-23 shows the size and parameter values for coding styles.

Lrgn: Length of marker segment in bytes (not including the marker).

Crgn: The number of the component to which this marker segment relates. The components are numbered 0, 1, 2, etc. (Either 8 or 16 bits depending on Csiz value.)

Srgn: ROI style for the current ROI. Table A-24 shows the value for the Srgn parameter.

SPrgn^i: Parameters for ROI style designated in Srgn. The parameters are designate, in order from top to bottom, in the appropriate table.

Table A-23 — Region-of-interest parameter values

Parameter	Size (bits)	Values
RGN	16	0xFF5E
Lrgn	16	4 — 65534
Crgn	8 16	0 — 255; if Csiz < 257 0 — 65534; Csiz ≥ 257
Srgn	8	use Table A-24
SPrgn^i	variable	refer to Table A-24

Table A-24 — Region-of-interest parameter values for the Srgn parameter

Values	ROI style (Srgn)	SPrgn usage
0	Implicit ROI (maximum shift)	use Table A-25
	All other values reserved	

Table A-25 — Region-of-interest values from SPrgn parameter (Srgn = 0)

Parameters (in order)	Size (bits)	Values	Meaning of SPrgn value
Implicit ROI shift	8 bits	0 — 255	Binary shifting of ROI coefficients above the background

A.6.4 Quantization default (QCD)

Function: Describes the quantization used for compressing all components. The parameter values can be overridden for an individual component by a QCC marker segment in either the main or tile-part header.

Usage: Shall be one and only one in the main header. May be one and only one in each tile-part headers. If there are multiple tile-parts in a tile, and this marker segment is present, it shall be found only in the first tile-part.

When used in the tile-part header it overrides the main QCD, main QCC, and tile QCD for the specific component. Thus, the hierarchy is the following:

Tile-part QCC > Tile-part QCD > Main QCC > Main QCD

Length: Variable depending on the number of quantized elements.

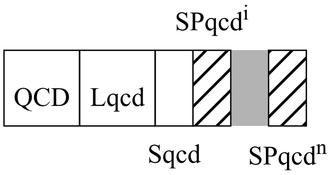


Figure A-10 — Quantization default syntax

QCD: Marker value. Table A-26 shows the size and parameter values for coding styles.

Lqcd: Length of marker segment in bytes (not including the marker).

Sqcd: Quantization style for all components.

SPqcd^i: Quantization step size value for the ith sub-band in the defined order (see Annex D.4). The number of parameters is the same as, or larger than, the number of sub-bands in the tile-component with the greatest number of decomposition levels.

Table A-26 — Quantization default parameter values

Parameter	Size (bits)	Values
QCD	16	0xFF5C
Lqcd	16	5 — 65534
Sqcd	8	use Table A-27
SPqcd^i	variable	refer to Table A-27

Table A-27 — Quantization default values for the Sqcd and Sqcc parameters

Values (bits) MSB LSB	Quantization style	SPqcx size	SPqcx usage
xxx0 0000	No quantization	8 bits	use Table A-28
xxx0 0001	Scalar implicit (values signalled for LL sub-band only)	16 bits	use Table A-29
xxx0 0010	Scalar explicit (values signalled for each sub-band)	16 bits	use Table A-29

Table A-27 — Quantization default values for the Sqcd and Sqcc parameters

Values (bits) MSB LSB	Quantization style	SPqcx size	SPqcx usage
000x xxxx — 111x xxxx	Number of guard bits 0 — 7		

Table A-28 — Reversible step size values for the SPqcd and SPqcc parameters

Values (bits) MSB LSB	Reversible step size values
xxx0 0000 — xxx1 1111	Exponent, ε_b, of the reversible dynamic range (repeated for every sub-band)

Table A-29 — Quantization values for scalar quantization for the SPqcd and SPqcc parameters

Values (bits) MSB LSB	Quantization step size values
xxxx x000 0000 0000 — xxxx x111 1111 1111	Mantissa, μ_b, of the quantization step size value (see Equation E.2)
0000 0xxx xxxx xxxx — 1111 1xxx xxxx xxxx	Exponent, ε_b, of the quantization step size value (see Equation E.2)

A.6.5 Quantization component (QCC)

Function: Describes the quantization used for compressing a particular component.

Usage: Optional in both the main and tile-part headers. No more than one per component may be present in either the main or tile-part headers. If there are multiple tile-parts in a tile, and this marker segment is present, it shall be found only in the first tile-part.

Optional in both the main and tile-part headers. When used in the main header it overrides the main QCD marker segment for the specific component. When used in the tile-part header it overrides the main QCD, main QCC, and tile QCD for the specific component. Thus, the hierarchy is the following:

Tile-part QCC > Tile-part QCD > Main QCC > Main QCD

No more one QCC marker segment per component may be used. It there are multiple tile-parts in a tile, then these marker segments shall be found only in the first tile-part.

Length: Variable depending on the number of quantized elements.

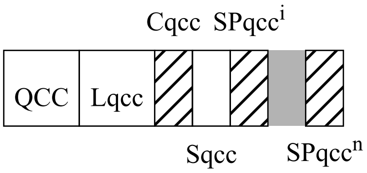


Figure A-11 — Quantization component syntax

QCC: Marker value. Table A-30 shows the size and parameter values for coding styles.

Lqcc: Length of marker segment in bytes (not including the marker).

Cqcc: The number of the component to which this marker segment relates. The components are numbered 0, 1, 2, etc. (Either 8 or 16 bits depending on Csiz value.)

Sqcc: Quantization style for this component.

SPqcc^i:Quantization value for each sub-band in the defined order (see Annex D.4). The number of parameters is the same as, or larger than, the number of sub-bands in the tile-component with the greatest number of decomposition levels.

Table A-30 — Quantization component parameter values

Parameter	Size (bits)	Values
QCC	16	0xFF5D
Lqcc	16	6 — 65534
Cqcc	8 16	0 — 255; if Csiz < 257 0 — 65534; Csiz ≥ 257
Sqcc	8	use Table A-27
SPqcc^i	variable	refer to Table A-27

A.6.6 Progression order change, main (POM)

Function: Describes the origin and progression order for any progression order changes in the tile.

Usage: Either POM or POT is required if there are progression order changes. May not be used when there are POT marker segments is used.

Length: Variable depending on the number of changes in the progression.

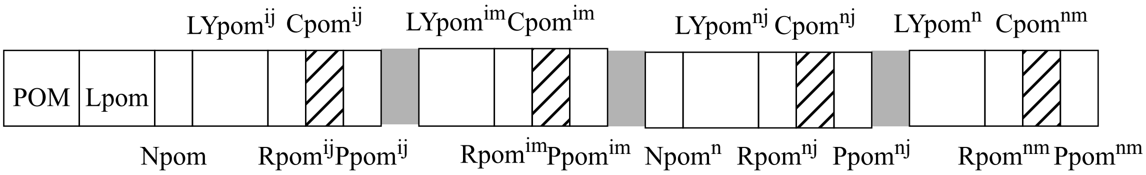


Figure A-12 — Progression order change, main syntax

POM: Marker value. Table A-32 shows the size and parameter values for coding styles.

Lpom:Length of marker segment in bytes (not including the marker).

Npom$^{\mathbf{i}}$:Number of progression changes for the ith tile in raster order in the codestream. There is one entry for each tile. Note that a zero value indicates there are no progression

LYpom$^{\mathbf{ij}}$:Layer index for the start of a new progression for the ith tile. One value for each progression change in each tile multiplied by the number of tiles.

Rpom$^{\mathbf{ij}}$:Resolution index for the start of a new progression for the ith tile. One value for each progression change in each tile multiplied by the number of tiles.

Cpom$^{\mathbf{ij}}$:Component index for the start of a new progression. The components are numbered 0, 1, 2, etc. (Either 8 or 16 bits depending on Csiz value.) One value for each progression change in each tile multiplied by the number of tiles.

Ppom$^{\mathbf{ij}}$:New progression order. One value for each progression change in each tile multiplied by the number of tiles.

Table A-31 — Progression order change, main parameter values

Parameter	Size (bits)	Values
POM	16	0xFF5E
Lpom	16	3 — 65534
Npom^{i}	8	0 — 255
LYpom^{ij}	16	0 — 65534
Rpom^{ij}	8	0 — 255
Cpom^{ij}	8 16	0 — 255; if Csiz < 257 0 — 65534; Csiz ≥ 257
Ppom^{ij}	8	use Table A-15

A.6.7 Progression order change, tile (POT)

Function: Describes the origin and progression order for any progression order changes in the tile.

Usage: Either POM or POT is required if there are progression order changes. May not be used when the POM marker segment is used.

Length: Variable depending on the number of changes in the progression.

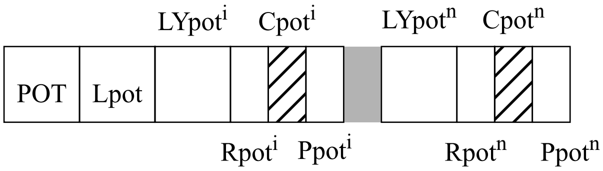


Figure A-13 — Progression order change, tile syntax

POT: Marker value. Table A-32 shows the size and parameter values for coding styles.

Lpot: Length of marker segment in bytes (not including the marker).

LYpot^i:Layer index for the start of a new progression. One value for each progression change in each tile.

Rpot^i: Resolution index for the start of a new progression. One value for each progression change in each tile.

Cpot^i: Component index for the start of a new progression. The components are numbered 0, 1, 2, etc. (Either 8 or 16 bits depending on Csiz value.) One value for each progression change in each tile.

Ppom^i:New progression order. One value for each progression change in each tile.

Table A-32 — Progression order change, tile parameter values

Parameter	Size (bits)	Values
POT	16	0xFF5F
Lpot	16	6 — 65534
LYpot^i	16	0 — 65534
Rpot^i	8	0 — 255
Cpot^i	8 16	0 — 255; if Csiz < 257 0 — 65534; Csiz ≥ 257
Ppot^i	8	use Table A-15

A.6.8 Error resilience style (ERS)

Function: Indicates that error resilience tools are present and which set of error resilience tools are used.

Usage: One marker segment only may be in the main header or any tile-part header. If present in the main header, it applies for all components of all tiles. If present in a tile header, it applies for all components in that tile and overrides an ERS in the main header.

Length: Fixed.

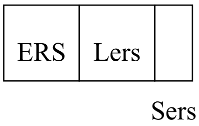


Figure A-14 — Error resilience bit stream syntax

ERS: Marker value. Table A-33 shows the size and parameter values for coding styles.

Lers: Length of marker segment in bytes (not including the marker).

Sers: Indicates which error resilience tools are present.

Table A-33 — Error resilience bit stream parameter values

Parameter	Size (bits)	Values
ERS	16	0xFF59
Lers	16	3
Sers	8	use Table A-34

Table A-34 — Error resilience bit stream tools for the Sers parameter

Values (bits) MSB LSB	Error resilience tools
xxxx xxx0 xxxx xxx1 xxxx xx0x xxxx xx1x	No SOP marker segments used SOP marker segments are used No segmentation symbols are used Segmentation symbols are used
	All other values reserved

A.7 Pointer marker segments

Pointer marker segments either provide a length or point into the codestream.The TLM marker segment describes the length of the tile-parts. It has the same length information as the SOT marker segment. The PLM or PLT marker segment describes the length of the packets in the bit stream of the packets.

NOTE — Having the pointer marker segments all occur in the main header allows direct access into the compressed data. Having the pointer information in the tile-part headers removes the burden on the encoder of rewinding to store the information.

The TLM (Ptlm) or the SOT (Psot) parameters point from the beginning of the current tile-part's SOT marker segment to the end of the data in that tile-part. Because tile-parts are required to be a multiple of 8 bits, these values are always a byte length. Figure A-15 shows the length of a tile-part.

The PLM or PLT marker segments are optional if all the components in a tile have the same progression style and number of layers specified in the appropriate COD marker segment (see Annex A.6.1). Either a PLM marker segment is used in the main header, or PLT marker segments are used in tile-part headers, or neither are used, but both shall not be used. The PLM and PLT marker segments are lengths of each packet in the tile-part.

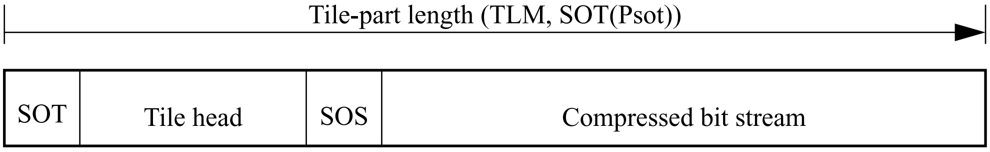


Figure A-15 — Tile-part lengths

A.7.1 Tile-part lengths, main header (TLM)

Function: Describes the length of every tile-part in the codestream. Each tile-part's length is measured from the first byte of the SOT marker segment to the end of the data of that tile-part. The value of each individual tile-part length in the TLM marker segment is the same as the value in the corresponding Psot in the SOT marker segment.

Usage: Optional use in the main header. There may be multiple TLM marker segment in the main header.

Length: Variable depending on the number of tile-parts in the image.

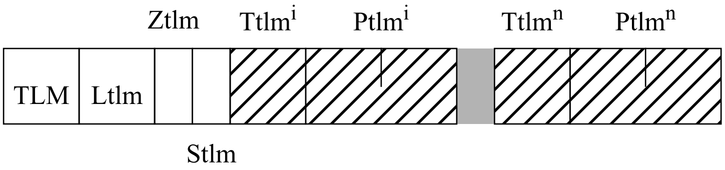


Figure A-16 — Tile-part length, main header syntax

TLM: Marker value. Table A-35 shows the size and values for the tile-part length main header parameters.

Ltlm: Length of marker segment in bytes (not including the marker).

Ztlm: Index of this marker segment relative to all other TLM markers present in the current header. For the full list of parameters that follow, the lists of every like marker segment are concatenated in order.

Stlm: Size of the Ttlm and Ptlm parameters.

Ttlm^i: Tile number of the ith tile-part. The Ttlm parameter is in the same order as the tile-parts the codestream. Either none or one value for every tile-part. The number of tile-parts can be derived from the length of this marker segment.

NOTE — The highest numerical values (255 or 65535) are reserved to indicate data other than tile data.

Ptlm^i: Length, in bytes, from the beginning of the SOT marker of the ith tile-part to the end of the data for that tile-part. One value for every tile-part. The number of tile-parts can be derived from the length of this marker segment.

Table A-35 — Tile-part length, main header parameter values

Parameter	Size (bits)	Values
TLM	16	0xFF55
Ltlm	16	5 — 65534
Ztlm	8	0 — 255
Stlm	8	use Table A-36
Ttlm^i	0 if ST = 0 8 if ST = 1 16 if ST = 2	tile in order 0 — 254 0 — 65334
Ptlm^i	16 if SP = 0 32 if SP = 1	2 — 65534 2 — (2^{32}-2)

Table A-36 — Size parameters for Stlm

Values (bits) MSB LSB	Parameter size
xx00 xxxx	ST = 0; Ttlm parameter is 0 bits, only one tile-part per tile, and tile-parts are in index order without omission or repetition
xx01 xxxx	ST = 1; Ttlm parameter 8 bits
xx10 xxxx	ST = 2; Ttlm parameter 16 bits
x0xx xxxx	SP = 0; Ptlm parameter 16 bits
x1xx xxxx	SP = 1; Ptlm parameter 32 bits
	All other values reserved

A.7.2 Packet length, main header (PLM)

Function: A list of packet lengths in the tile-parts. This exists for every tile-part in order.

Usage: Can be used only in the main header, if there are packets in the file and an PLT marker segment is not used. There may be multiple PLM marker segments. Both the PLM and PLT marker segments are optional.

Length: Variable depending on the number of tile-parts in the image and the number of packets in each tile-part.

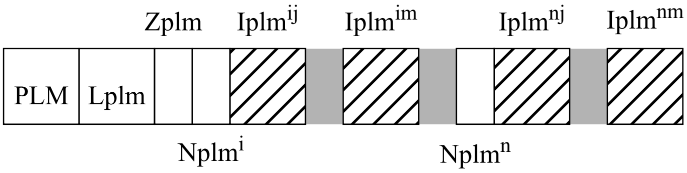


Figure A-17 — Packets length, main header syntax

PLM: Marker value. Table A-37 shows the size and values for the packets, main header parameters.

Lplm: Length of marker segment in bytes (not including the marker).

Zplm: Index of this marker segment relative to all other PLM markers present in the current header. For the full list of parameters that follow, the lists of every like marker segment are concatenated in order.

Nplm^i: Number of bytes of Iplm information for the ith tile-part in the order found in the codestream. One value for each tile.

Iplm^{ij}: Length of the jth packet in the ith tile-part. If packet header is stored with the packet this length includes the packet header, if packet headers are stored in PPM or PPT this length does not include the packet header length. One range of values for each tile-part. One value for each packet in the tile.

Table A-37 — Packets length, main header parameter values

Parameter	Size (bits)	Values
PLM	16	0xFF57
Lplm	16	4 — 65534
Zplm	8	0 — 255
Nplm^i	8	0 — 255
Iplm^{ij}	variable	use Table A-38

Table A-38 — Iplm, Iplt list of lengths for regular progression of packets

Parameters (in order)	Size (bits)	Values	Meaning of Iplm or Iplt values
Packet length	8 bits repeated as necessary	0xxx xxxx 1xxx xxxx x000 0000 — x111 1111	Last 7 bits of packet length, terminate number[a] Continue reading[b] 7 bits of packet length

a. This is the last 7 bits that make up the packet length.

b. This is not the last 7 bits that make up the packet length. Note that each 7 bit portion of the packet length is read from the codestream and packed MSB first to make up the length of the packet.

A.7.3 Packets length, tile-part header (PLT)

Function: A list of packet lengths in the tile-part.

Usage: Can be used in any tile header, if there are packets in the file and an PLM marker segment is not used. There may be multiple PLT marker segments per tile. The PLT marker is found only in the first tile-part. Both the PLM and PLT marker segments are optional.

Length: Variable depending on the number of packets in each tile-part.

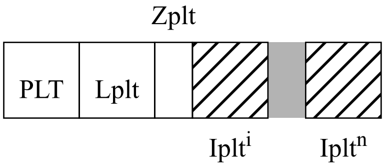


Figure A-18 — Packet length, tile header syntax

PLT: Marker value. Table A-39 shows the size and values for the packet parameters.

Lplt: Length of marker segment in bytes (not including the marker).

Zplt: Index of this marker segment relative to all other PLT markers present in the current header. For the full list of parameters that follow, the lists of every like marker segment are concatenated in order.

Iplm^i: Length of the ith packet. If packet headers are stored with the packet this length includes the packet header, if packet headers are stored in PPM or PPT this length does not include the packet header length. One value for each packet in the tile.

Table A-39 — Packet length, tile-part headers parameter values

Parameter	Size (bits)	Values
PLT	16	0xFF58
Lplt	16	3 — 65534
Zplt	8	0 — 255
Iplt^i	variable	refer to Table A-38

A.7.4 Packed packet headers, main header (PPM)

Function: A collection of the packet headers so multiple reads are not required to decode headers.

Usage: May be used in the main header for all tile-parts with packets unless a PPT marker is used in the tile header.

The packet headers shall be in only one of three places though the codestream. If the PPM marker segment is present, all the packet headers are found in that marker segment. In this case, the PPT marker segment and packets distributed in the bit stream of the tile-parts are disallowed.

If there is no PPM marker segment than the packets can be distributed either in a PPT marker segment in the first tile-part or distributed in the in the codestream as defined in Annex D.7. The packet headers shall not be in both a PPT marker segment and the codestream for the same tile. there may be multiple PPM marker segments in this header.

Length: Variable depending on the number of packets in each tile-part and the compression of the packet headers.

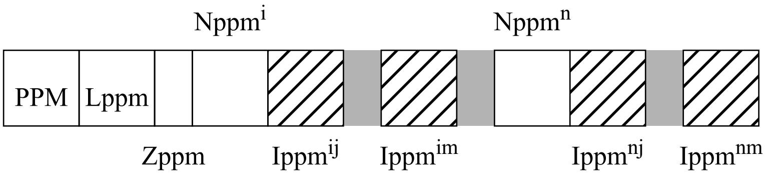


Figure A-19 — Packed packet headers, main header syntax

PPM: Marker value. Table A-40 shows the size and values for the parameters.

Lppm:Length of marker segment in bytes, not including the marker.

Zppm:Index of this marker segment relative to all other PPM markers present in the current header. For the full list of parameters that follow, the lists of every like marker segment are concatenated in order.

$Nppm^i$: Number bytes of Iplm information for the ith tile-part in the order found in the codestream. One value for each tile.

$Ippm^{ij}$: Packet header for every packet in order in the tile-part. The component number, layer, and resolution are determined from the method of progression or the POM or POT marker. The contents are exactly the packet header which would have been distributed in the bit stream as described in Annex D.5 packet header information. One range of values for each tile-part. One value for each packet in each tile.

Table A-40 — Packed packet headers, main header parameter values

Parameter	Size (bits)	Values
PPM	16	0xFF60
Lppm	16	8 — 65535
Zppm	8	0 — 255
$Nppm^i$	16	0 - 65535
$Ippm^{ij}$	variable	packet headers

A.7.5 Packed packet headers, tile-part header (PPT)

Function: A collection of the packet headers so multiple reads are not required to decode headers.

Usage: May be used in the every tile-part header for a tile-part with packets unless a PPM marker is used in the main header.

The packet headers shall be in only one of three places though the codestream. If the PPM marker segment is present, all the packet headers are found in that marker segment. In this case, the PPT marker segment and packets distributed in the bit stream of the tile-parts are disallowed.

If there is no PPM marker segment than the packets can be distributed either in a PPT marker segment in the first tile-part or distributed in the in the codestream as defined in Annex D.7. The packet headers shall not be in both a PPT marker segment and the codestream for the same tile. There may be multiple of the PPT marker segment in a tile-part header.

Length: Variable depending on the number of packets in each tile-part and the compression of the packet headers.

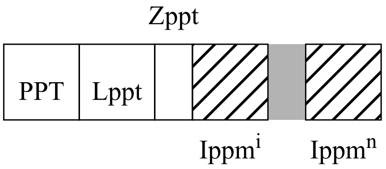


Figure A-20 — Packed packet headers, tile header syntax

PPT: Marker value. Table A-41 shows the size and values for the parameters.

Lppt: Length of marker segment in bytes, not including the marker.

Zppt: Index of this marker segment relative to all other PPT markers present in the current header. For the full list of parameters that follow, the lists of every like marker segment are concatenated in order.

Ippt^i: Packet header for every packet in order in the tile-part. The component number, layer, and resolution are determined from the method of progression or the POM or POT marker. The contents are exactly the packet header which would have been distributed in the bit stream as described Annex D.5 packet header information. One value for each packet in the tile.

Table A-41 — Packet header, tile headers parameter values

Parameter	Size (bits)	Values
PPT	16	0xFF61
Lppt	16	3 — 65535
Zppt	8	0 — 255
Ippt^i	variable	packet headers

A.8 In bit stream marker segments

These marker segments are used for error resilience. The marker segments are different because they have no length field and they are found in the bit stream, not the main or a tile-part header.

A.8.1 Start of partition (SOP)

Function: Marks the beginning of a partition and the index of that partition within a codestream.

Usage: Optional. Using in the bit stream in front of every packet. The first packet in a tile are assigned

Length: Fixed 4 bytes.

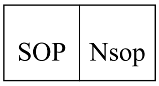


Figure A-21 — Start of partition syntax

SOP: Marker value. Table A-42 shows the size and values for start of tile-part.

Nsop: Packet sequence number. The first packet in a tile is assigned the value zero. For every successive packet this number is incremented by one. When the maximum number is reached, the number rolls over to zero.

Table A-42 — Start of partition parameter values

Parameter	Size (bits)	Values
SOP	16	0xFF91
Nsop	16	0 — 65535

A.9 Informational markers

These marker segments are strictly information and are not necessary for a decoder. However, these marker segments might assist a parser or decoder. More information about the source and characteristics of the image can be obtained by using a file format such as SPIFF, TIFF, or JP2 (see Annex J).

A.9.1 Comment and extension (CME)

Function: Allows unstructured data in the header.

Usage: Repeatable as many times as desired in either or both the main or tile-part headers.

Length: Variable depending on the length of the message.

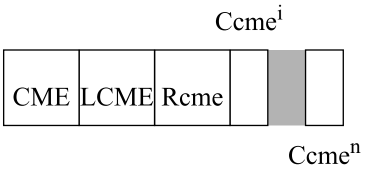


Figure A-22 — Coding style component syntax

CME: Marker value. Table A-43 shows the size and values for the comment parameters.

Lcme: Length of marker segment in bytes (not including the marker).

Rcme: Registration value of the marker segment.

Ccme^i:Byte of unstructured data.

Table A-43 — Comment and extension parameter values

Parameter	Size (bits)	Values
CME	16	0xFF64
Lcme	16	5 — 65534
Rcme	16	use Table A-44
Ccme^i	8	0 — 255

Table A-44 — Registration values for the Rcme parameter

Values	Registration values
0	General use (binary values)
1	General use (ISO 8859-1 (latin-1) values)
2 —_65534	Reserved for registration
65535	Reserved for extension

Annex B

Arithmetic entropy coding

(This annex forms an integral part of this Recommendation | International Standard)

This annex defines the lossless arithmetic entropy coding. This annex is compatible with the arithmetic coder defined in ITU-T Rec.T.87 | ISO/IEC 14496-1.

In this Annex and all of its subclauses, the flow charts and tables are normative only in the sense that they are defining an output that alternative implementations shall duplicate.

B.1 Binary encoding (informative)

Figure B-1 shows a simple block diagram of the binary adaptive arithmetic encoder. The decision (D) and context (CX) pairs are processed together to produce compressed data (CD) output. Both D and CX are provided by the model unit (not shown). CX selects the probability estimate to use during the coding of D. In this International Standard, CX is a label for a context.

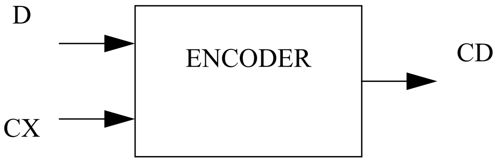


Figure B-1 — Arithmetic encoder inputs and outputs

B.1.1 Recursive interval subdivision (informative)

The recursive probability interval subdivision of Elias coding is the basis for the binary arithmetic coding process. With each binary decision the current probability interval is subdivided into two sub-intervals, and the code string is modified (if necessary) so that it points to the base (the lower bound) of the probability sub-interval assigned to the symbol which occurred.

In the partitioning of the current interval into two sub-intervals, the sub-interval for the more probable symbol (MPS) is ordered above the sub-interval for the less probable symbol (LPS). Therefore, when the MPS is coded, the LPS sub-interval is added to the code string. This coding convention requires that symbols be recognized as either MPS or LPS, rather than 0 or 1. Consequently, the size of the LPS interval and the sense of the MPS for each decision must be known in order to code that decision.

Since the code string always points to the base of the current interval, the decoding process is a matter of determining, for each decision, which sub-interval is pointed to by the compressed data. This is also done recursively, using the same interval sub-division process as in the encoder. Each time a decision is decoded, the decoder subtracts any interval the encoder added to the code string. Therefore, the code string in the decoder is a pointer into the current interval relative to the base of the current interval. Since the coding process involves addition of binary fractions rather than concatenation of integer code words, the more probable binary decisions can often be coded at a cost of much less than one bit per decision.

B.1.2 Coding conventions and approximations (informative)

The coding operations are done using fixed precision integer arithmetic and using an integer representation of fractional values in which 0x8000 is equivalent to decimal 0.75. The interval A is kept in the range $0.75 \leq A < 1.5$ by doubling it whenever the integer value falls below 0x8000.

The code register C is also doubled each time A is doubled. Periodically – to keep C from overflowing – a byte of data is removed from the high order bits of the C-register and placed in an external compressed data buffer. Carry-over into the external buffer is resolved by a bit stuffing procedure.

Keeping A in the range $0.75 \leq A < 1.5$ allows a simple arithmetic approximation to be used in the interval subdivision. The interval is A and the current estimate of the LPS probability is Qe, a precise calculation of the sub-intervals would require:

$$A - (Qe * A) = \text{sub-interval for the MPS} \qquad \text{B.1}$$

$$Qe * A = \text{sub-interval for the LPS} \qquad \text{B.2}$$

Because the value of A is of order unity, these are approximated by

$$A - Qe = \text{sub-interval for the MPS} \qquad \text{B.3}$$

$$Qe = \text{sub-interval for the LPS} \qquad \text{B.4}$$

Whenever the MPS is coded, the value of Qe is added to the code register and the interval is reduced to A - Qe. Whenever the LPS is coded, the code register is left unchanged and the interval is reduced to Qe. The precision range required for A is then restored, if necessary, by renormalization of both A and C.

With the process illustrated above, the approximations in the interval subdivision process can sometimes make the LPS sub-interval larger than the MPS sub-interval. If, for example, the value of Qe is 0.5 and A is at the minimum allowed value of 0.75, the approximate scaling gives 1/3 of the interval to the MPS and 2/3 to the LPS. To avoid this size inversion, the MPS and LPS intervals are exchanged whenever the LPS interval is larger than the MPS interval. This MPS/LPS conditional exchange can only occur when a renormalization is needed.

Whenever a renormalization occurs, a probability estimation process is invoked which determines a new probability estimate for the context currently being coded. No explicit symbol counts are needed for the estimation. The relative probabilities of renormalization after coding an LPS and MPS provide an approximate symbol counting mechanism which is used to directly estimate the probabilities.

B.2 Description of the arithmetic encoder (informative)

The ENCODER (Figure B-2) initializes the encoder through the INITENC procedure. CX and D pairs are read and passed on to ENCODE until all pairs have been read. The probability estimation procedures which provide adaptive estimates of the probability for each context are imbedded in ENCODE. Bytes of compressed data are output when no longer modifiable. When all of the CX and D pairs have been read (Finished?), FLUSH sets the contents of the C-register to as many 1-bits as possible and then outputs the final bytes. FLUSH also terminates the encoding and generates the required terminating marker.

NOTE — While FLUSH is required in ITU-T Rec.T.87 | ISO/IEC 14496-1 it is informative in this specification. Other methods are acceptable.

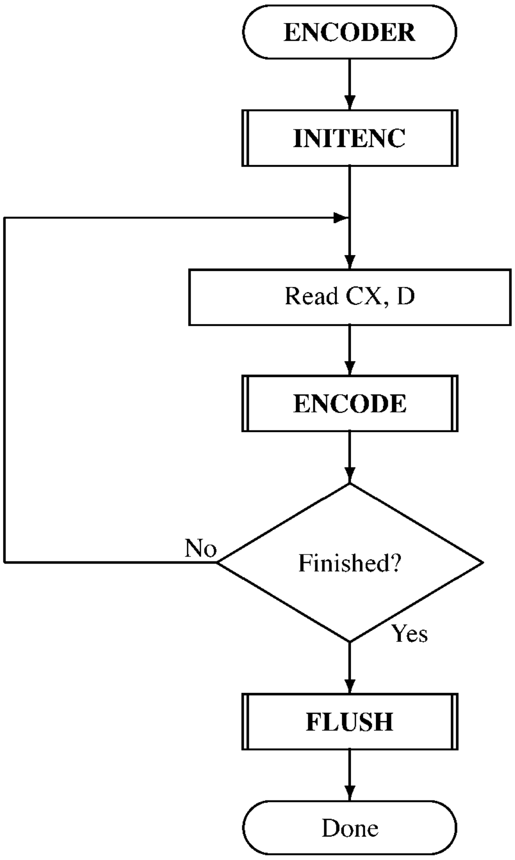


Figure B-2 — Encoder for the MQ-coder

B.2.1 Encoder code register conventions (informative)

The flow charts given in this Annex assume the register structures for the encoder shown in Table B-1.

Table B-1 — Encoder register structures

	MSB			LSB
C-register	0000 cbbb	bbbb bsss	xxxx xxxx	xxxx xxxx
A-register	0000 0000	0000 0000	aaaa aaaa	aaaa aaaa

The “a” bits are the fractional bits in the A-register (the current interval value) and the “x” bits are the fractional bits in the code register. The “s” bits are spacer bits which provide useful constraints on carry-over, and the “b” bits indicate the bit positions from which the completed bytes of the data are removed from the C-register. The “c” bit is a carry bit.

The detailed description of bit stuffing and the handling of carry-over will be given in a later part of this Annex.

B.2.2 Encoding a decision (ENCODE) (informative)

The ENCODE procedure determines whether the decision D is a 0 or not. Then a CODE0 or a CODE1 procedure is called appropriately. Often embodiments will not have an ENCODE procedure, but will call the CODE0 or CODE1 procedures directly to code a 0-decision or a 1-decision. Figure B-3 shows this procedure.

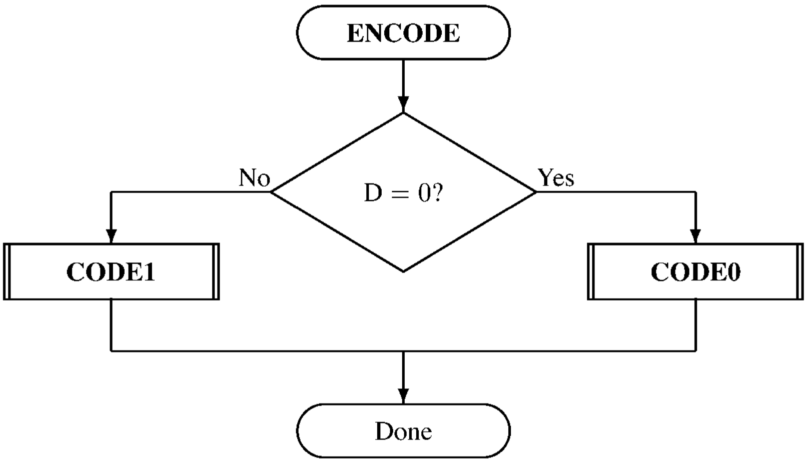


Figure B-3 — ENCODE procedure

B.2.3 Encoding a 1 or a 0 (CODE1 and CODE0) (informative)

When a given binary decision is coded, one of two possibilities occurs – the symbol is either the more probable symbol or it is the less probable symbol. CODE1 and CODE0 are illustrated in Figure B-4 and Figure B-5. In these figures, CX is the context. For each context, the index of the probability estimate which is to be used in the coding operations and the MPS value are stored. MPS(CX) is the sense (0 or 1) of the MPS for context CX.

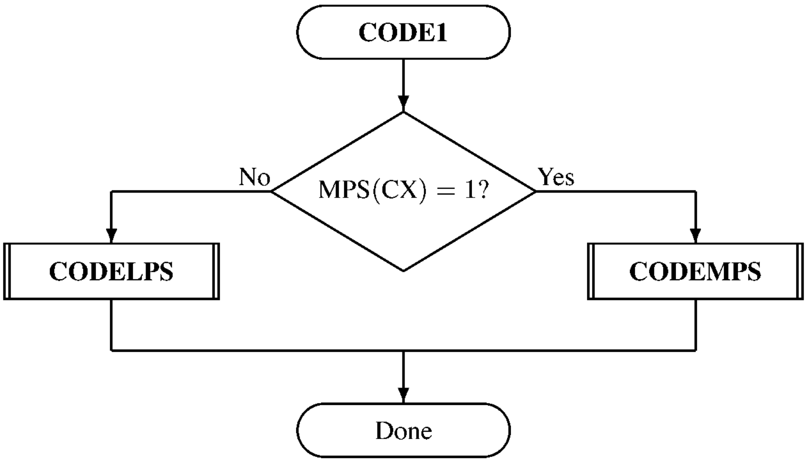


Figure B-4 — CODE1 procedure

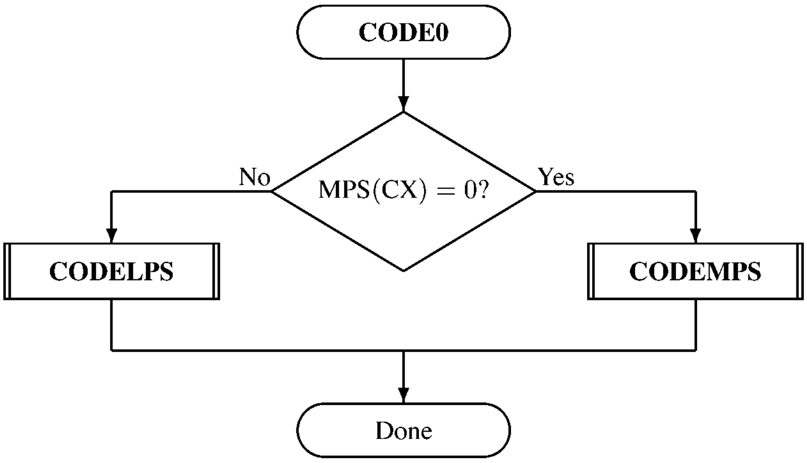


Figure B-5 — CODE0 procedure

B.2.4 Encoding an MPS or LPS (CODEMPS and CODELPS)

The CODELPS (Figure B-6) procedure usually consists of a scaling of the interval to Qe(I(CX)), the probability estimate of the LPS determined from the index I stored for context CX. The upper interval is first calculated so it can be compared to the lower interval to confirm that Qe has the smaller size. It is always followed by a renormalization (RENORME). In the event that the interval sizes are inverted, however, the conditional MPS/LPS exchange occurs and the upper interval is coded. In either case, the probability estimate is updated. If the SWITCH flag for the index I(CX) is set, then the MPS(CX) is inverted. A new index I is saved at CX as determined from the next LPS index (NLPS) column in Table B-2.

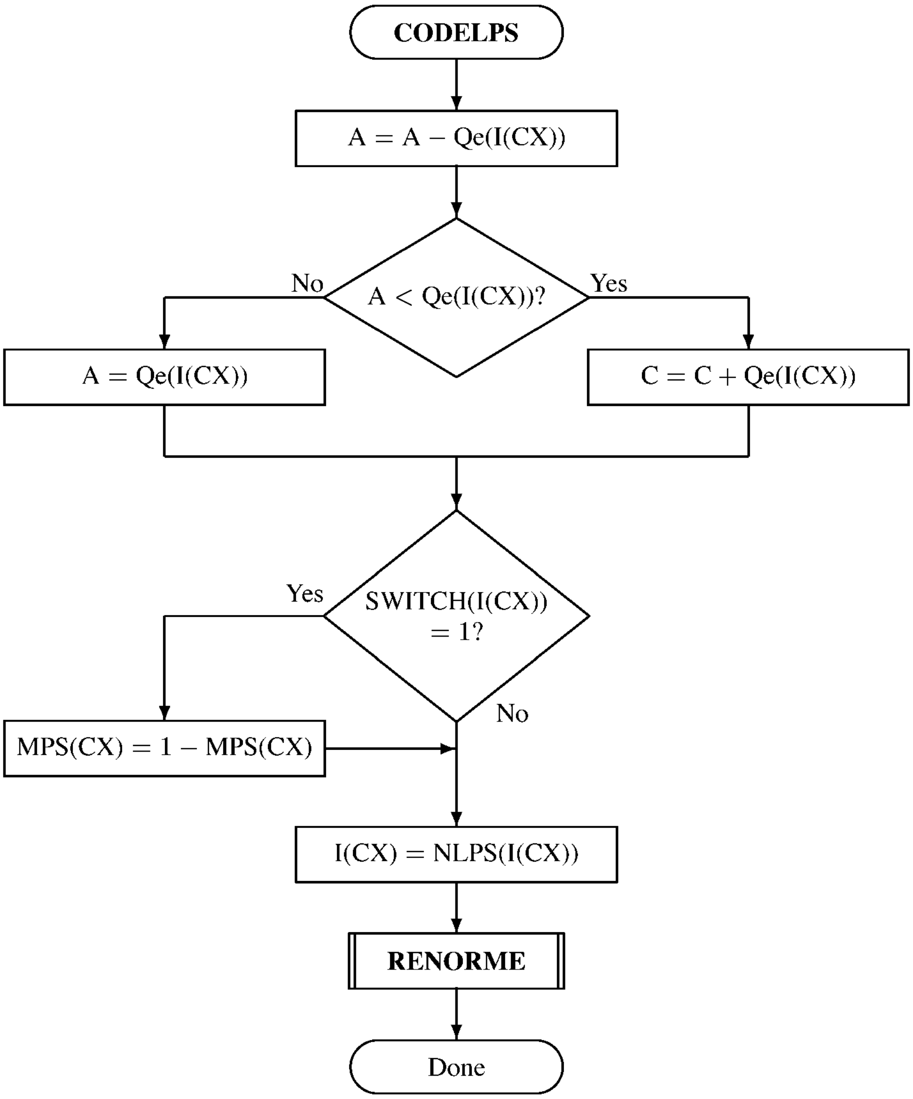


Figure B-6 — CODELPS procedure with conditional MPS/LPS exchange

Table B-2 — Qe values and probability estimation process

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
0	0x5601	0101 0110 0000 0001	0.503937	1	1	1
1	0x3401	0011 0100 0000 0001	0.304715	2	6	0
2	0x1801	0001 1000 0000 0001	0.140650	3	9	0
3	0x0ACl	0000 1010 1100 0001	0.063012	4	12	0

Table B-2 — Qe values and probability estimation process

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
4	0x0521	0000 0101 0010 0001	0.030053	5	29	0
5	0x0221	0000 0010 0010 0001	0.012474	38	33	0
6	0x5601	0101 0110 0000 0001	0.503937	7	6	1
7	0x5401	0101 0100 0000 0001	0.492218	8	14	0
8	0x4801	0100 1000 0000 0001	0.421904	9	14	0
9	0x3801	0011 1000 0000 0001	0.328153	10	14	0
10	0x3001	0011 0000 0000 0001	0.281277	11	17	0
11	0x2401	0010 0100 0000 0001	0.210964	12	18	0
12	0x1C01	0001 1100 0000 0001	0.164088	13	20	0
13	0x1601	0001 0110 0000 0001	0.128931	29	21	0
14	0x5601	0101 0110 0000 0001	0.503937	15	14	1
15	0x5401	0101 0100 0000 0001	0.492218	16	14	0
16	0x5101	0101 0001 0000 0001	0.474640	17	15	0
17	0x4801	0100 1000 0000 0001	0.421904	18	16	0
18	0x3801	0011 1000 0000 0001	0.328153	19	17	0
19	0x3401	0011 0100 0000 0001	0.304715	20	18	0
20	0x3001	0011 0000 0000 0001	0.281277	21	19	0
21	0x2801	0010 1000 0000 0001	0.234401	22	19	0
22	0x2401	0010 0100 0000 0001	0.210964	23	20	0
23	0x2201	0010 0010 0000 0001	0.199245	24	21	0
24	0x1C01	0001 1100 0000 0001	0.164088	25	22	0
25	0x1801	0001 1000 0000 0001	0.140650	26	23	0
26	0x1601	0001 0110 0000 0001	0.128931	27	24	0
27	0x1401	0001 0100 0000 0001	0.117212	28	25	0
28	0x1201	0001 0010 0000 0001	0.105493	29	26	0

Table B-2 — Qe values and probability estimation process

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
29	0x1101	0001 0001 0000 0001	0.099634	30	27	0
30	0x0ACl	0000 1010 1100 0001	0.063012	31	28	0
31	0x09C1	0000 1001 1100 0001	0.057153	32	29	0
32	0x08A1	0000 1000 1010 0001	0.050561	33	30	0
33	0x0521	0000 0101 0010 0001	0.030053	34	31	0
34	0x0441	0000 0100 0100 0001	0.024926	35	32	0
35	0x02Al	0000 0010 1010 0001	0.015404	36	33	0
36	0x0221	0000 0010 0010 0001	0.012474	37	34	0
37	0x0141	0000 0001 0100 0001	0.007347	38	35	0
38	0x0111	0000 0001 0001 0001	0.006249	39	36	0
39	0x0085	0000 0000 1000 0101	0.003044	40	37	0
40	0x0049	0000 0000 0100 1001	0.001671	41	38	0
41	0x0025	0000 0000 0010 0101	0.000847	42	39	0
42	0x0015	0000 0000 0001 0101	0.000481	43	40	0
43	0x0009	0000 0000 0000 1001	0.000206	44	41	0
44	0x0005	0000 0000 0000 0101	0.000114	45	42	0
45	0x0001	0000 0000 0000 0001	0.000023	45	43	0
46	0x5601	0101 0110 0000 0001	0.503937	46	46	0

The CODEMPS (Figure B-7) procedure usually reduces the size of the interval to the MPS sub-interval and adjusts the code register so that it points to the base of the MPS sub-interval. However, if the interval sizes are inverted, the LPS sub-interval is coded instead. Note that the size inversion cannot occur unless a renormalization (RENORME) is required after the coding of the symbol. The probability estimate update changes the index I(CX) according to the next MPS index (NMPS) column in Table B-2.

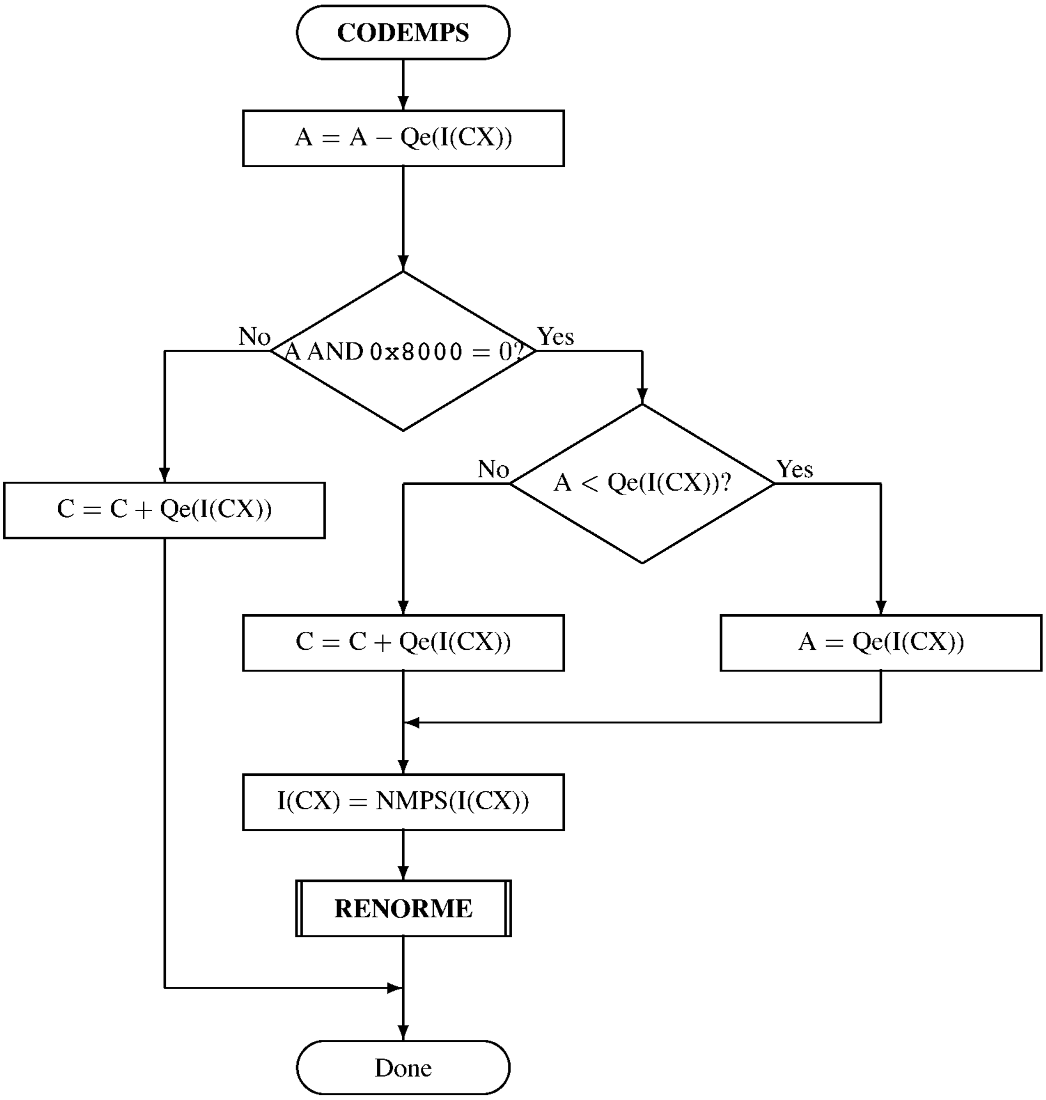


Figure B-7 — CODEMPS procedure with conditional MPS/LPS exchange

B.2.5 Probability Estimation

Table B-2 shows the Qe value associated with each Qe index. The Qe values are expressed as hexadecimal integers, as binary integers, and as decimal fractions. To convert the 15 bit integer representation of Qe to the decimal probability, the Qe values are divided by (4/3) * (0x8000).

The estimator can be defined as a finite-state machine – a table of Qe indexes and associated next states for each type of renormalization (i.e., new table positions) – as shown in Table B-2. The change in state occurs only when the arithmetic coder interval register is renormalized. This is always done after coding the LPS, and whenever the interval register is less than 0x8000 (0.75 in decimal notation) after coding the MPS.

After an LPS renormalization, NLPS gives the new index for the LPS probability estimate. After an MPS renormalization, NMPS gives the new index for the LPS probability estimate. If Switch is 1, the MPS symbol sense is reversed.

The index to the current estimate is part of the information stored for context CX. This index is used as the index to the table of values in NMPS, which gives the next index for an MPS renormalization. This index is saved in the context storage at CX. MPS(CX) does not change.

The procedure for estimating the probability on the LPS renormalization path is similar to that of an MPS renormalization, except that when SWITCH(I(CX)) is 1, the sense of MPS(CX) is inverted.

The final index state 46 can be used to establish a fixed 0.5 probability estimate.

B.2.6 Renormalization in the encoder (RENORME) (informative)

Renormalization is very similar in both encoder and decoder, except that in the encoder it generates compressed bits and in the decoder it consumes compressed bits.

The RENORME procedure for the encoder renormalization is illustrated in Figure B-8. Both the interval register A and the code register C are shifted, one bit at a time. The number of shifts is counted in the counter CT, and when CT is counted down to zero, a byte of compressed data is removed from C by the procedure BYTEOUT. Renormalization continues until A is no longer less than 0x8000.

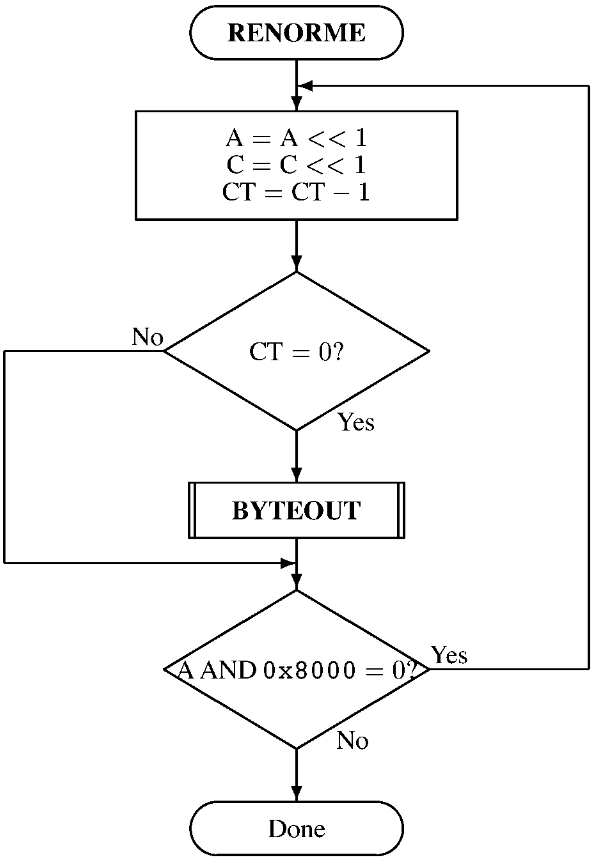


Figure B-8 — Encoder renormalisation procedure

B.2.7 Compressed data output (BYTEOUT) (informative)

The BYTEOUT routine called from RENORME is illustrated in Figure B-9. This routine contains the bit-stuffing procedures which are needed to limit carry propagation into the completed bytes of compressed data. The conventions used make it impossible for a carry to propagate through more than the byte most recently written to the compressed data buffer.

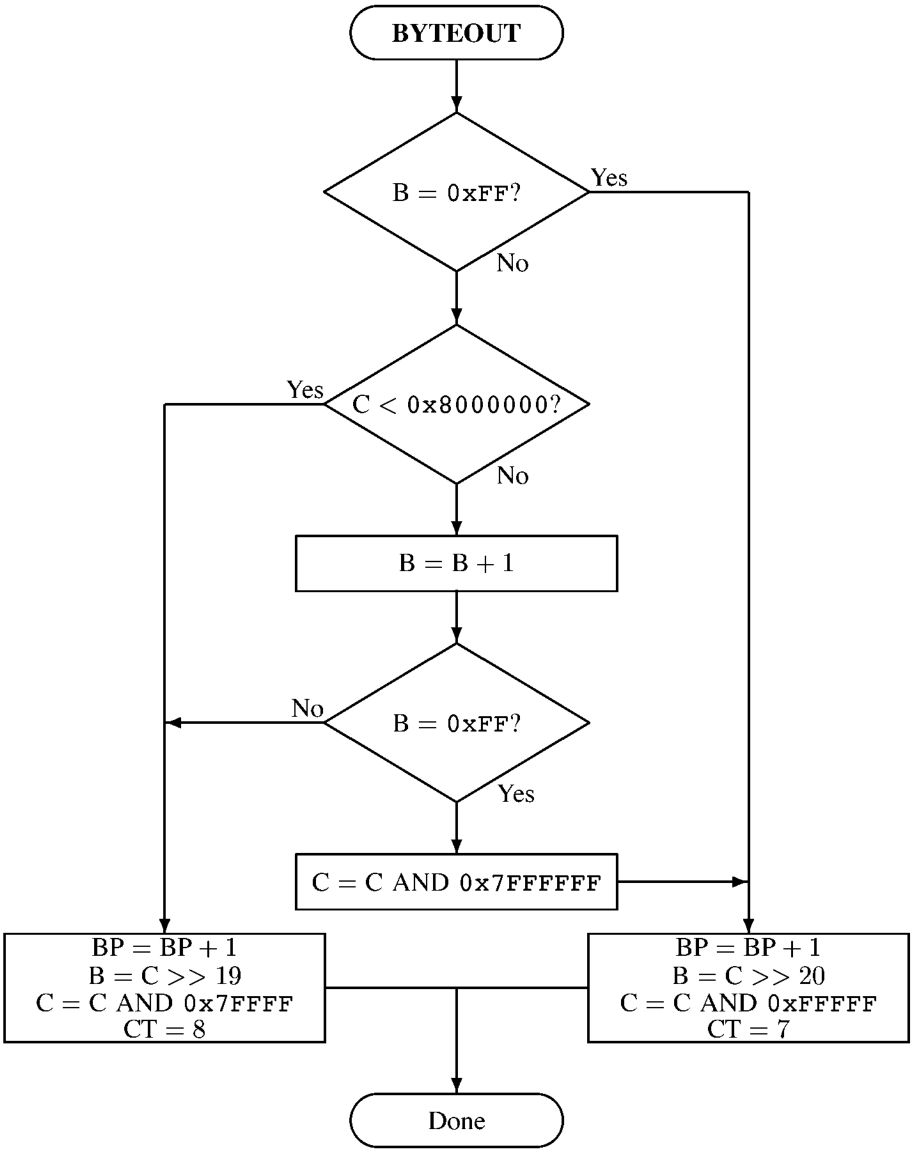


Figure B-9 — BYTEOUT procedure for encoder

The procedure in the block in the lower right section does bit stuffing after a 0xFF byte; the similar procedure on the left is for the case where bit stuffing is not needed.

B is the byte pointed to by the compressed data buffer pointer BP. If B is not a 0xFF byte, the carry bit is checked. If the carry bit is set, it is added to B and B is again checked to see if a bit needs to be stuffed in the next byte. After the need for bit stuffing has been determined, the appropriate path is chosen, BP is incremented and the new value of B is removed from the code register “b” bits.

B.2.8 Initialisation of the encoder (INITENC) (informative)

The INITENC procedure is used to start the arithmetic coder. The basic steps are shown in Figure B-10.

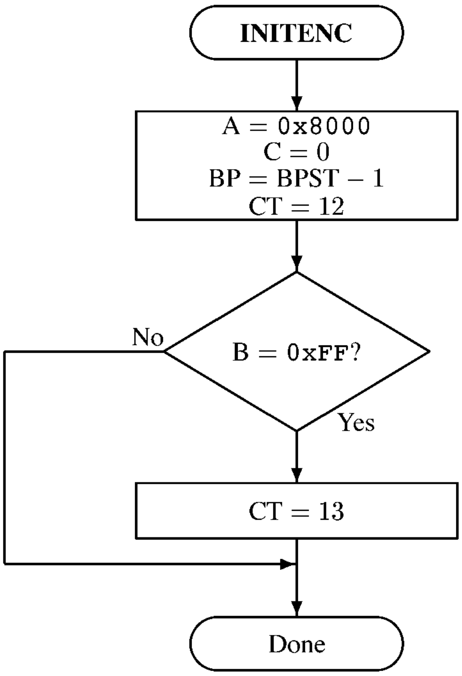


Figure B-10 — Initialisation of the encoder

The interval register and code register are set to their initial values, and the bit counter is set. Setting CT = 12 reflects the fact that there are three spacer bits in the register which need to be filled before the field from which the bytes are removed is reached. Note that BP always points to the byte preceding the position BPST where the first byte is placed. Therefore, if the preceding byte is a 0xFF byte, a spurious bit stuff will occur, but can be compensated for by increasing CT. The default settings for MPS and I are shown in Table C-6.

B.2.9 Termination of coding (FLUSH) (informative)

The FLUSH procedure shown in Figure B-11 is used to terminate the encoding operations and generate the required terminating marker. The procedure guarantees that the 0xFF prefix to the marker code overlaps the final bits of the compressed data. This guarantees that any marker code at the end of the compressed data will be recognized and interpreted before decoding is complete.

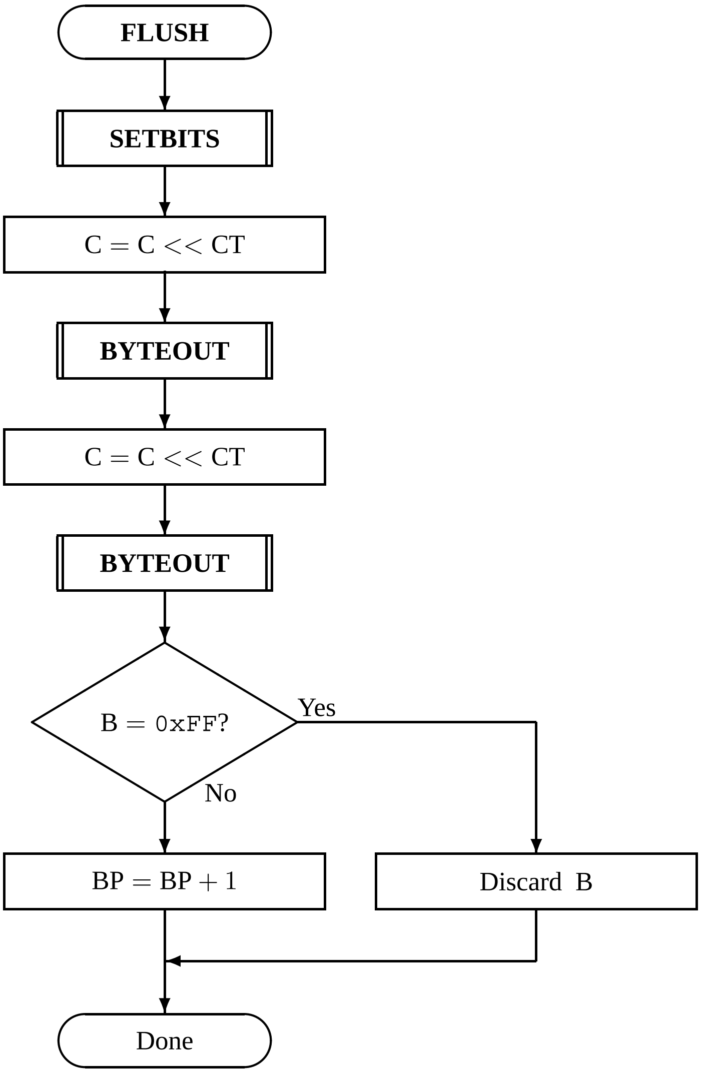


Figure B-11 — FLUSH procedure

The first part of the FLUSH procedure sets as many bits in the C-register to 1 as possible as shown in Figure B-12. The exclusive upper bound for the C-register is the sum of the C-register and the interval register. The low order 16 bits of C are forced to 1, and the result is compared to the upper bound. If C is too big, the leading 1-bit is removed, reducing C to a value which is within the interval.

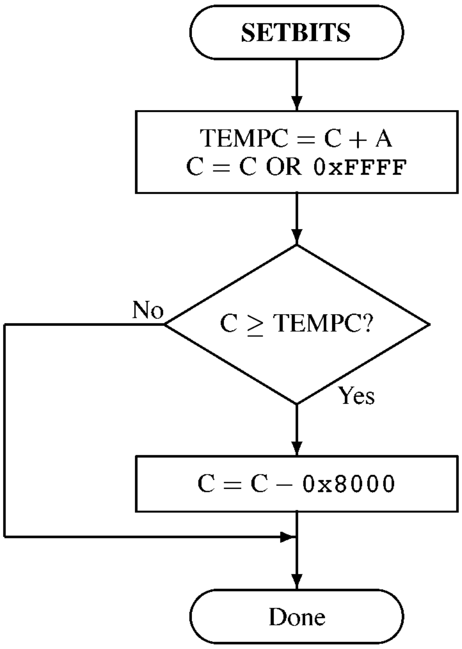


Figure B-12 — Setting the final bits in the C register

The byte in the C-register is then completed by shifting C, and two bytes are then removed. If the byte in buffer, B, an 0xFF then it is discarded. Otherwise, buffer B is output to the bit stream.

NOTE — This is the only normative option for termination in ITU-T Rec.T.87 | ISO/IEC 14496. However, further reduction of the bit stream is allowed provided correct decoding is assured (see Annex C.5.2).

B.3 Arithmetic decoding procedure

Figure B-13 shows a simple block diagram of a binary adaptive arithmetic decoder. The compressed data CD and a context CX from the decoder's model unit (not shown) are input to the arithmetic decoder. The decoder's output is the decision D. The encoder and decoder model units need to supply exactly the same context CX for each given decision.

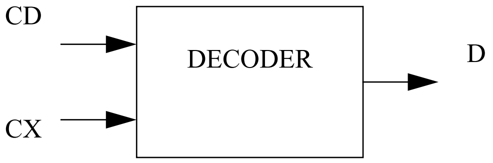


Figure B-13 — Arithmetic decoder inputs and outputs

The DECODER (Figure B-14) initializes the decoder through INITDEC. Contexts, CX, and bytes of compressed data (as needed) are read and passed on to DECODE until all contexts have been read. The DECODE routine decodes the binary decision D and returns a value of either 0 or 1. The probability estimation procedures which provide adaptive estimates of the probability for each context are embedded in DECODE. When all contexts have been read (Finished?), the compressed data has been decompressed.

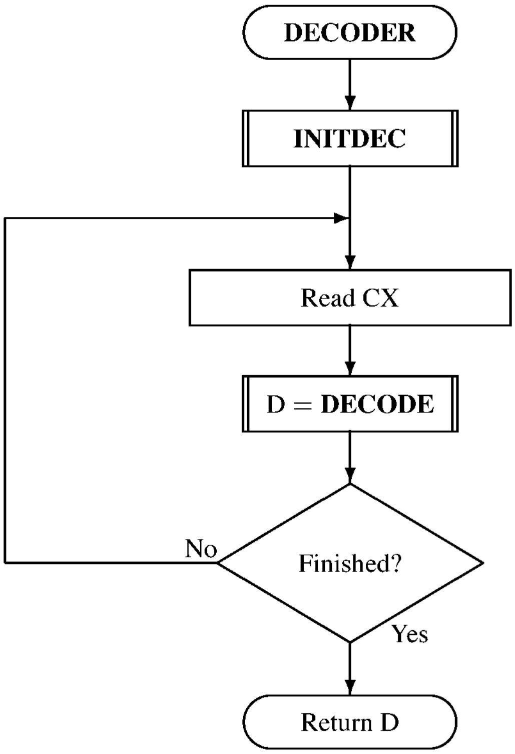


Figure B-14 — Decoder for the MQ-coder

B.3.1 Decoder code register conventions

The flow charts given in this Annex assume the register structures for the decoder shown in Table B-3.

Table B-3 — Decoder register structures

	MSB	LSB
Chigh register	xxxx xxxx	xxxx xxxx
Clow register	bbbb bbbb	0000 0000
A-register	aaaa aaaa	aaaa aaaa

Chigh and Clow can be thought of as one 32 bit C-register in that renormalization of C shifts a bit of new data from the MSB of Clow to the LSB of Chigh. However, the decoding comparisons use Chigh alone. New data is inserted into the "b" bits of Clow one byte at a time.

The detailed description of the handling of data with stuff-bits will be given later in this Annex.

Note that the comparisons shown in the various procedures in this section assume precisions greater than 16 bits. Logical comparisons can be used with 16 bit precision.

B.3.2 Decoding a decision (DECODE)

The decoder decodes one binary decision at a time. After decoding the decision, the decoder subtracts any amount from the compressed data that the encoder added. The amount left in the compressed data is the offset from the base of the current interval to the sub-interval allocated to all binary decisions not yet decoded. In the first test in the DECODE procedure illustrated in Figure B-15 the Chigh register is compared to the size of the LPS sub-interval. Unless a conditional exchange is needed, this test determines whether a MPS or LPS is decoded. If Chigh is logically greater than or equal to the LPS probability estimate Qe for the current index I stored at CX, then Chigh is decremented by that amount. If A is not less than 0x8000, then the MPS sense stored at CX is used to set the decoded decision D.

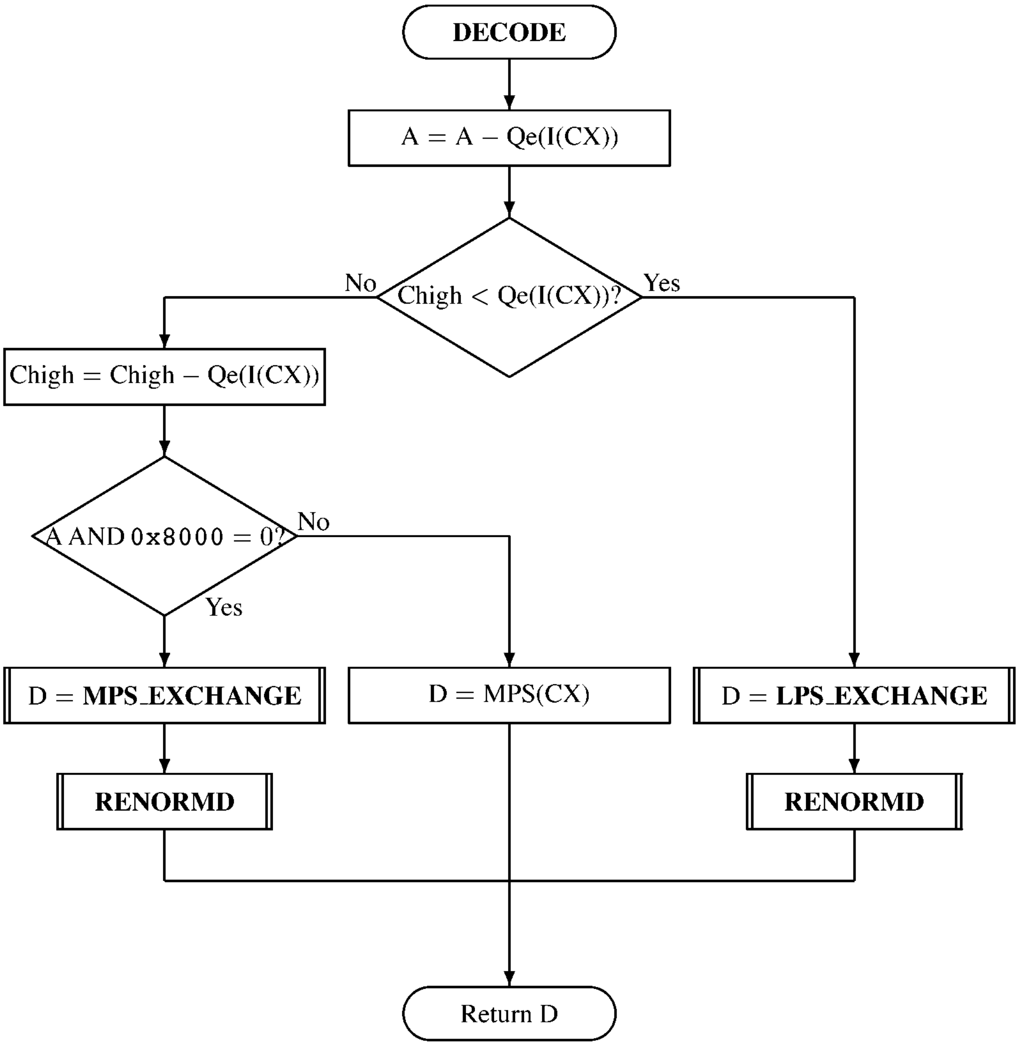


Figure B-15 — Decoding an MPS or an LPS

When a renormalization is needed, the MPS/LPS conditional exchange may have occurred. For the MPS path the conditional exchange procedure is shown in Figure B-16. As long as the MPS sub-interval size A calculated as the first step in Figure B-16 is not logically less than the LPS probability estimate Qe(I(CX)), an MPS did occur and the decision can be set from MPS(CX). Then the index I(CX) is updated from the next MPS index (NMPS) column in Table B-2. If, however, the LPS sub-interval is larger, the conditional exchange occurred and an LPS occurred. The probability update switches the MPS sense if the SWITCH column has a "1" and updates the index I(CX) from the next LPS index (NLPS) column in Table B-2. Note that the probability estimation in the decoder needs to identical to the probability estimation in the encoder.

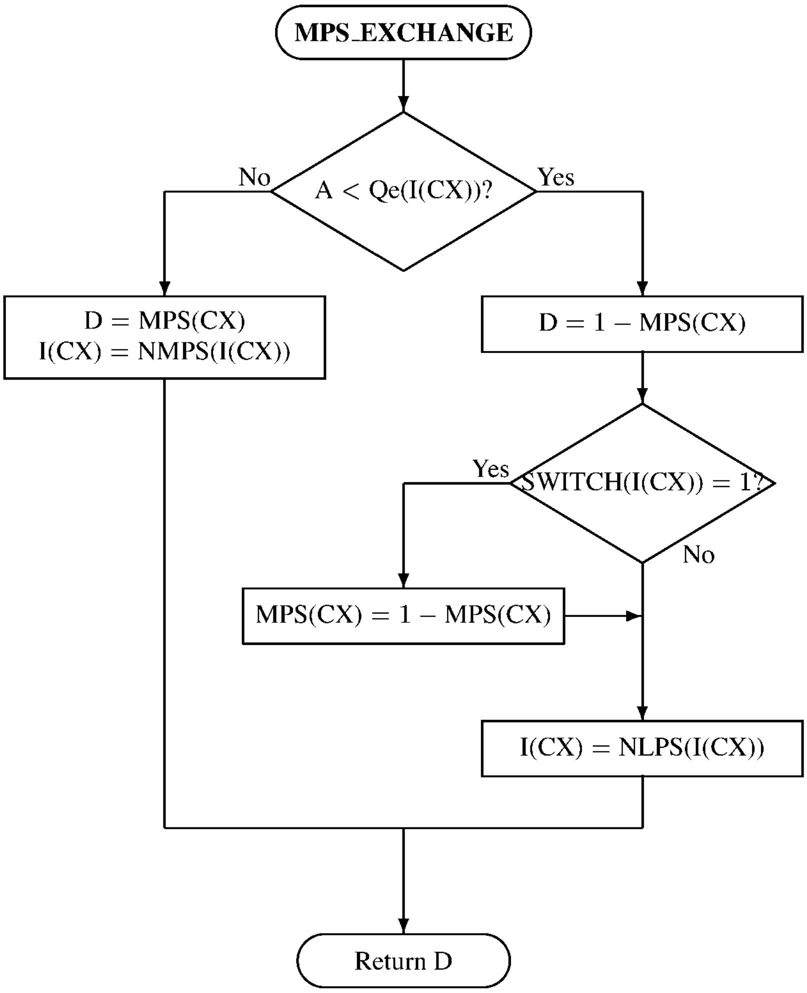


Figure B-16 — Decoder MPS path conditional exchange procedure

For the LPS path of the decoder the conditional exchange procedure is given the LPS_EXCHANGE procedure shown in Figure B-17. The same logical comparison between the MPS sub-interval A and the LPS sub-interval Qe(I(CX)) determines if a conditional exchange occurred. On both paths the new sub-interval A is set to Qe(I(CX)). On the left path the conditional exchange occurred so the decision and update are for the MPS case. On the right path, the LPS decision and update are followed.

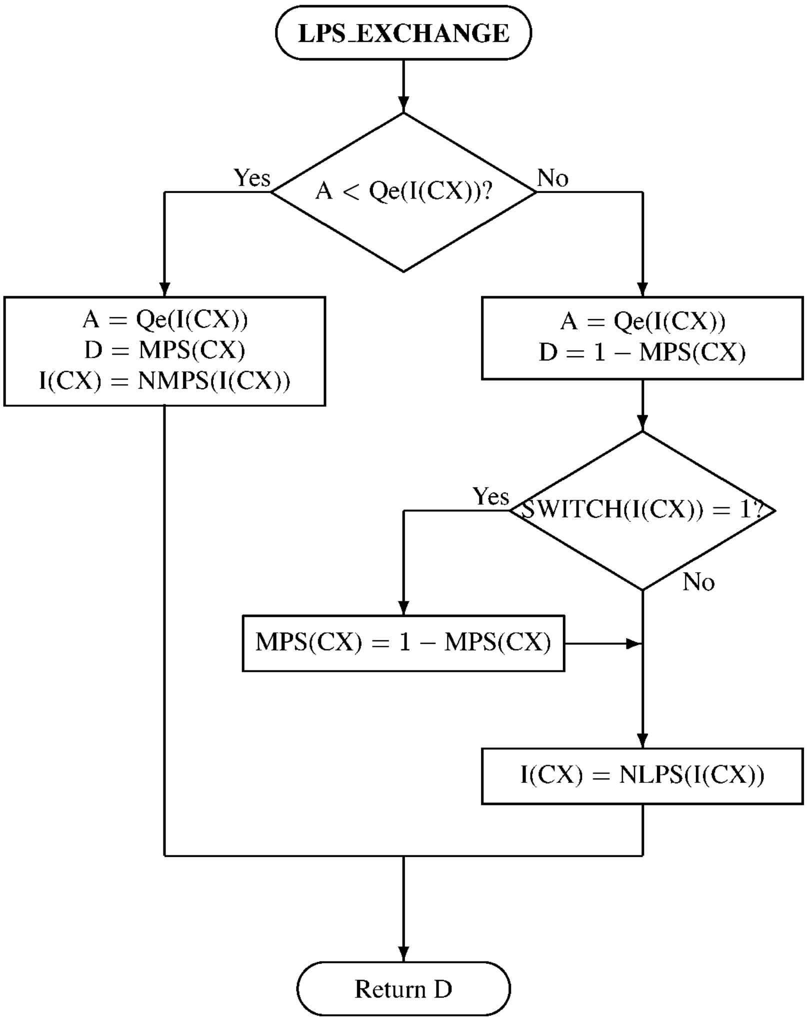


Figure B-17 — Decoder LPS path conditional exchange procedure

B.3.3 Renormalization in the decoder (RENORMD)

The RENORMD procedure for the decoder renormalization is illustrated in Figure B-18. A counter keeps track of the number of compressed bits in the Clow section of the C-register. When CT is zero, a new byte is inserted into Clow in the BYTEIN procedure.

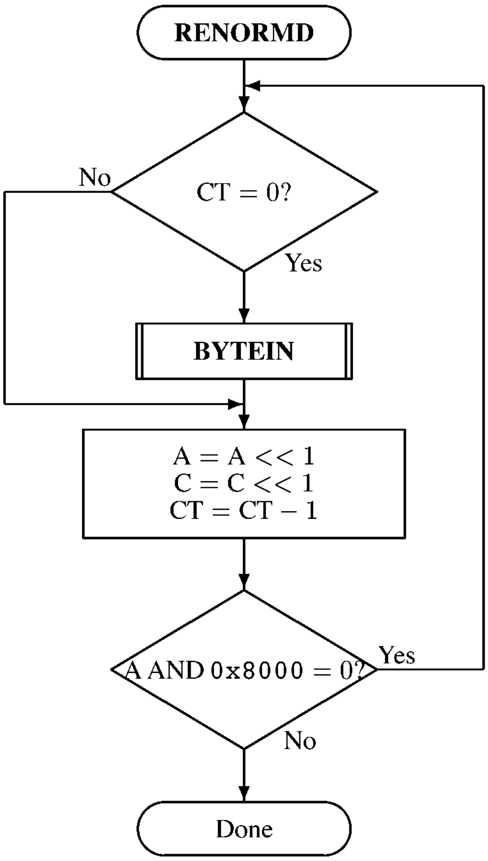


Figure B-18 — Decoder renormalisation procedure

Both the interval register A and the code register C are shifted, one bit at a time, until A is no longer less than 0x8000.

B.3.4 Compressed data input (BYTEIN)

The BYTEIN procedure called from RENORMD is illustrated in Figure B-19. This procedure reads in one byte of data, compensating for any stuff bits following the 0xFF byte in the process. It also detects the marker codes which must occur at the end of a scan or resynchronization interval. The C-register in this procedure is the concatenation of the Chigh and Clow registers.

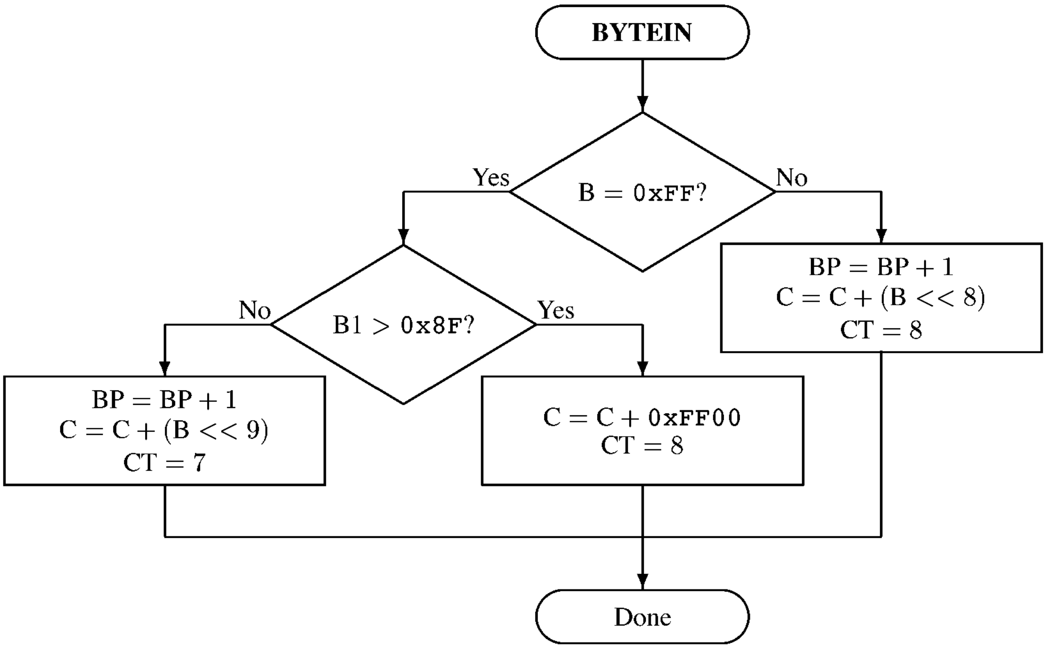


Figure B-19 — BYTEIN procedure for decoder

B is the byte pointed to by the compressed data buffer pointer BP. If B is not a 0xFF byte, BP is incremented and the new value of B is inserted into the high order 8 bits of Clow.

If B is a 0xFF byte, then B1 (the byte pointed to by BP+1) is tested. If B1 exceeds 0x8F, then B1 must be one of the marker codes. The marker code is interpreted as required, and the buffer pointer remains pointed to the 0xFF prefix of the marker code which terminates the arithmetically compressed data. 1-bits are then fed to the decoder until the decoding is complete. This is shown by adding 0xFF00 to the C-register and setting the bit counter CT to 8.

If B1 is not a marker code, then BP is incremented to point to the next byte which contains a stuffed bit. The B is added to the C-register with an alignment such that the stuff bit (which contains any carry) is added to the low order bit of Chigh.

B.3.5 Initialisation of the decoder (INITDEC)

The INITDEC procedure is used to start the arithmetic decoder. The basic steps are shown in Figure B-20.

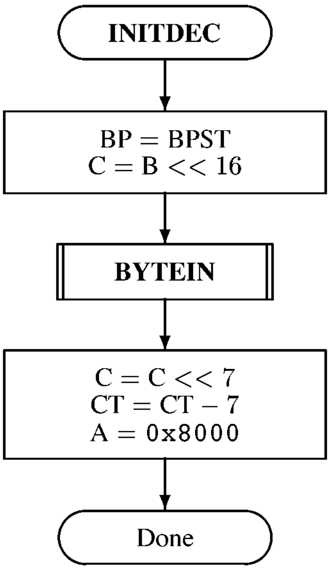


Figure B-20 — Initialisation of the decoder

BP, the pointer to the compressed data, is initialized to BPST (pointing to the first compressed byte). The first byte of the compressed data is shifted into the low order byte of Chigh, and a new byte is then read in. The C-register is then shifted by 7 bits and CT is decremented by 7, bringing the C-register into alignment with the starting value of A. The interval register A is set to match the starting value in the encoder.

B.3.6 Resetting arithmetic coding statistics

At certain points during the decoding some or all of the arithmetic coding statistics are reset. This process involves setting I(CX) and MPS(CX) equal to zero for some or all values of CX.

B.3.7 Saving arithmetic coding statistics

In some cases, the decoder needs to save or restore some values of I(CX) and MPS(CX).

Annex C

Coefficient bit modeling

(This annex forms an integral part of this Recommendation | International Standard)

This annex defines the modeling of the transform coefficient bits. It describes how the coefficients are arranged into code-blocks, bit-planes, and coding passes.

The coefficients are associated with different sub-bands arising from the transform applied (see Annex F). These coefficients are then arranged into rectangular blocks within each sub-band, called code-blocks. These code-blocks are then coded a bit-plane at a time starting with the most significant bit-plane with a non-zero element to the least significant bit-plane.

For each bit-plane in a code-block, a special code-block scan pattern is used for each of three passes. Each coefficient bit in the bit-plane is coded in only one of the three passes. The passes are called significance propagation, magnitude refinement, and cleanup. For each pass contexts are created which is passed to the arithmetic coder, CX, along with the bit stream, CD, (see Annex B.3). The arithmetic coder is reset according to selected rules.

C.1 Division of the frequency sub-bands into code-blocks

As described in Annex F, all the transform coefficients $J_{t,c}(u,v)$ of an image tile component or of a cell of an image tile component are grouped into sub-bands to form sub-band coefficients $a_b(u,v)$ for each sub-band, b. The sub-bands are further partitioned into rectangular code-blocks. The basic size of these code-blocks is signalled in the COD or COC markers (see Annex A.6.1 and Annex A.6.2) and is the same for all sub-bands. With the possible exception is code-blocks on the boundary of sub-bands, all code-blocks have the same width ($2^{xcb'}$) and height ($2^{ycb'}$) for a given tile component. The code-block width and height are limited to powers of two with the minimum size being 2^2 and the maximum being 2^{10}. Further the code-block size is restricted to the xcb+ycb <= 12.

The code-block size is further constrained by the partition size. This is shown in Equation D.14 and Equation D.15.

A sub-band of coefficients $a_b(x,y)$ is partitioned into code-blocks in the following way. Figure C-1 shows the partition as defined by the width and height of the code-block. All first rows of code-blocks are located at $x = (m * 2^{xcb'})$ and all first columns of code-blocks are located at $y = (m * 2^{ycb'})$.

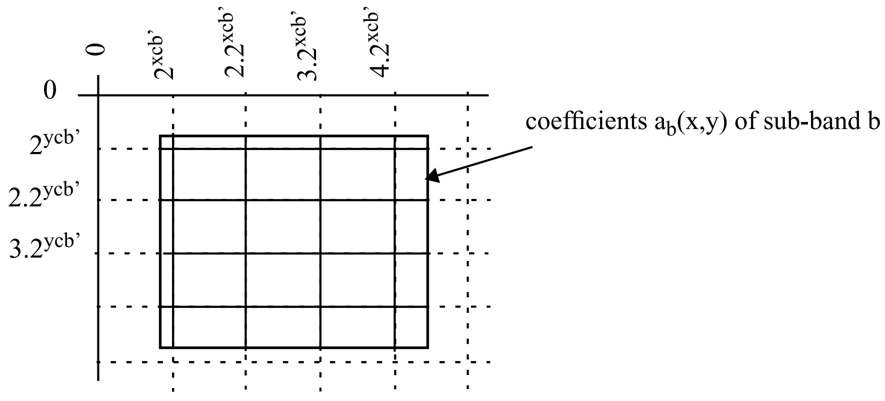
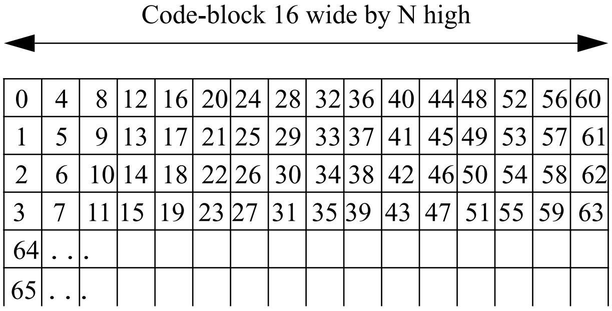


Figure C-1 — Partition of sub-band $a_b(x,y)$

C.2 Code-block scan pattern within code-blocks

Each bit-plane of a code-block is scanned in a particular order. Starting at the top left, the first four bits of the column are scanned. Then the first four bits of the second column, until the width of the code-block has been covered. Then the



0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
64	. . .														
65	. . .														

Figure C-2 — Example code-block scan pattern of a code-block

second four bits of the first column are scanned and so on. A similar vertical scan is continued for any leftover rows on the lowest code-blocks in the sub-band. Figure C-2 shows an example of the code-block scan pattern for a code-block.

C.3 Coefficient bits and significance

All quantized transform coefficients are signed values even when the original components are unsigned. These coefficients are expressed in a sign-magnitude representation prior to coding. For a particular sub-band, there is a maximum number of magnitude bits, M_b. The interesting point in a coefficient is the most significant 1 value magnitude bit. This is where the significance state will change (see the section below). For a code-block, the number of bit-planes from the most significant bit-plane that are all zero, is signalled in the packet header (see Annex D.7.5). No other coding of those insignificant bit-planes is made.

C.4 Decoding passes over the bit-planes

Each coefficient in a code-block has an associated binary state variable called its significance state. Significance states are initialized to 0 (coefficient is insignificant) and may become 1 (coefficient is significant) during the course of the coding of the code-block. The context vector for a given current coefficient is the binary vector consisting of the significance states of its 8 nearest-neighbor coefficients, as shown in Figure C-3. Any nearest neighbor lying outside the current coefficient's code-block is regarded as insignificant (i.e., it is treated as having a zero significance state) for the purpose of coding a bit in the current coefficient.

In general, a current coefficient can have 256 possible context vectors. These are clustered into a smaller number of contexts according to the rules specified below for context formation. Four different context formation rules are defined, one for each of the four coding operations: significance coding, sign coding, magnitude refinement coding, and cleanup coding. These coding operations are performed in three coding passes over each bit plane: significance and sign coding in a significance propagation, magnitude refinement coding in a magnitude refinement pass, and cleanup and sign coding in a cleanup pass. For a given coding operation, the context label (or context) provided to the arithmetic coding engine is a label assigned to the current coefficient's context.

NOTE — Although (for the sake of concreteness) specific integers are used in the tables below for labeling contexts, the tokens used for context labels are implementation-dependent and their values are not mandated by this Recommendation | International Standard.

The number of bit-planes from the most significant bit that have no significant coefficients (only insignificant bits) are signalled in the packet headers (see Annex D.7.5). The first bit-plane with a non-zero element has a cleanup pass only. The remaining bit-planes are coded in three passes. Each coefficient bit is coded only once in one of the three passes. Which pass a coefficient bit is coded in depends on the conditions for that pass. In general, the significance propagation includes the coefficients that are predicted, or "most likely," to become significant and their sign bits, as appropriate. The magnitude refinement pass includes bits from already significant coefficients. And the cleanup pass includes all the remaining coefficients.

D_0	V_0	D_1
H_0	“X”	H_1
D_2	V_1	D_3

Figure C-3 — Neighbors states used to form the context

C.4.1 Significance propagation decoding pass

The eight surrounding neighbor coefficients of a current coefficient (shown as a X in Figure C-3 where X denotes the current coefficient) are used to create 9 context bins based on how many and which ones are significant. If a coefficient is significant then it is given a 1 value for the creation of the context, otherwise it is given a 0 value. The mapping to the contexts also depends on which sub-band (at a given decomposition level) the code-block is in. Table C-1 shows these contexts.

Table C-1 — Contexts for the significance propagation and cleanup passes

LL and LH sub-bands (vertical high-pass)			HL sub-band (horizontal high-pass)			HH sub-band (diagonally high-pass)		Context label[a]
ΣH	ΣV	ΣD	ΣH	ΣV	ΣD	Σ(H+V)	ΣD	
2	x[b]	x	x	2	x	x	≥3	8
1	≥1	x	≥1	1	x	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5
0	2	x	2	0	x	1	1	4
0	1	x	1	0	x	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

a. Note that the context labels are numbered only for identification. The actual identifiers used is a matter of implementation.

b. x = do not care.

The significance propagation includes only bits of coefficients that were insignificant (the significance bit has yet to be encountered) and have a non-zero context. All other coefficients are skipped. The context is delivered to the arithmetic decoder (along with the bit stream) and the decoded coefficient bit is returned is returned. If the value this bit is 1 then the significance state is set to 1 and the immediate next bit to be decoded is the sign bit for the coefficient. Otherwise, the significance state remains 0. When the contexts of successive coefficients and passes are considered, the most current significance state for this coefficient is used.

C.4.2 Sign bit decoding

The context label for sign bit decoding is determined using another context of the neighborhood. Computation of the context label can be viewed as a two step process. The first step summarizes the contribution of the vertical and the horizontal neighbors. The second step reduces those contributions to one of 5 context labels.

For the first step, the two vertical neighbors (see Figure C-3) are considered together. Each neighbor may have one of three states: significant positive, significant negative, or insignificant. If the two vertical neighbors are both significant with the same sign, or if only one is significant then the vertical contribution is 1 if positive or -1 if negative. If both vertical neighbors are insignificant, or both are significant with different signs, then the vertical contribution is 0. The horizontal contribution is created the same way. Once again, if the neighbors fall outside the code-block they are considered to be insignificant. Table C-2 shows these contributions.

Table C-2 — Contributions of the vertical (and the horizontal) neighbors to the sign context

V_0 (or H_0)	V_1 (or H_1)	V (or H) contribution
significant, positive	significant, positive	1
significant, negative	significant, positive	0
insignificant	significant, positive	1
significant, positive	significant, negative	0
significant, negative	significant, negative	-1
insignificant	significant, negative	-1
significant, positive	insignificant	1
significant, negative	insignificant	-1
insignificant	insignificant	0

The second step reduces the nine permutations of the vertical and horizontal contributions into 5 context labels. Table C-3 shows these context labels. This context is delivered to the arithmetic decoder with the bit stream. The bit returned is then XORed with the XORbit in Table C-3 to produce the sign bit. The following equation is used:

$$signbit = AC(contextlabel) \oplus XORbit \qquad \text{C.1}$$

where signbit is the sign bit of the current coefficient, AC(contextlabel) is the value returned from the arithmetic decoder given the context label and the bit stream, and the XORbit is found in Table C-3 for the current context label.

Table C-3 — Sign contexts from the vertical and horizontal contributions

Horizontal contribution	Vertical contribution	XORbit	Context label
1	1	0	13
1	0	0	12
1	-1	0	11

Table C-3 — Sign contexts from the vertical and horizontal contributions

Horizontal contribution	Vertical contribution	XORbit	Context label
0	1	0	10
0	0	0	9
0	-1	1	10
-1	1	1	11
-1	0	1	12
-1	-1	1	13

C.4.3 Magnitude refinement pass

The magnitude refinement pass through the data includes the bits from coefficients that are already significant (except those that have just become significant in the immediately proceeding significance propagation).

The context used is determined by the summation of the significance state of the horizontal, vertical, and diagonal neighbors. Further, it is dependent on whether this is the first refinement bit (the bit immediately after the significance and sign bits) or not. Table C-4 shows the three context bins for this pass.

Table C-4 — Contexts for the magnitude refinement passes

$\Sigma H + \Sigma V$	First refinement for this coefficient	Context label
x[a]	false	16
≥1	true	15
0	true	14

[a] "x" indicates a "don’t care" state.

C.4.4 Cleanup pass

All the remaining coefficients are insignificant and had the context value of zero during the significance propagation. These are all included in the cleanup pass. The cleanup pass not only uses the neighbor context, like the significance propagation, from Table C-1, but also a run-length context.

First, the neighbor context for the coefficients in this pass are recreated using Table C-1. Note that the context label can now have any value because the coefficients that were found to be significant in the significance propagation are used as significant in the cleanup pass. Run-lengths are decoded with a unique single context. If the four contiguous coefficients in the column being scanned are all coded in the cleanup pass and the context label for all is 0 then the unique run-length context is given to the arithmetic decoder along with the bit stream. If the symbol, 0, is returned then all four contiguous coefficients in the column remain insignificant.

Otherwise, if a symbol, 1, is returned then at least one of the four contiguous coefficients in the column are significant. The next two bits, returned with the UNIFORM context (index 46 in Table B-2) denote which coefficient from the top of the column down is the first to be found significant. That coefficient’s sign bit is determined as described in Annex C.4.2. The decoding of any remaining coefficients continues in the manner described in Annex C.4.1.

If the four contiguous coefficients in a column are not all decoded in the cleanup pass or the context bin for any is non-zero, then the coefficient bits are decoded with the context in Table C-1 as in the significance propagation. Note that the same contexts as the significance propagation are used here (the state is used as well as the model). Table C-5 shows the logic for the cleanup pass.

Table C-5 — Run-length coder for cleanup decoding passes

Four contiguous coefficients in a column remaining to be decoded and each currently have the 0 context	Symbols with run-length context, 17	Four contiguous bits to be coded are zero	Symbols decoded with UNIFORM[a] context (	Number of coefficients to decode
true	0	true	none	none
true	1	false skip to first coefficient sign skip to second coefficient sign skip to third coefficient sign skip to fourth coefficient sign	 00 01 10 11	 3 2 1 0
false	none	x	none	rest of column

a. See Annex B.

If there are fewer than four rows remaining in a block, then no run-length coding is used. Once again, the significance state of any coefficient is changed immediately after decoding the first 1 magnitude bit.

C.5 Initializing and terminating

When the contexts are initialized, or re-initialized, they are set to the values in the Table C-6. The contexts are either re-initalized at the end of every coding pass, or only at the end of every code-block. Which is used is signaled in the COD or COC marker (see Annex A.6.1 and Annex A.6.2).

Table C-6 — Initial states for all 18 contexts

Context	Initial index from Table B-2	MPS
UNIFORM	46	0
Run-length	3	0
All zero neighbors (context label 0 in Table C-1)	4	0
All other contexts	0	0

In the normal operation (not selective arithmetic coding bypass), the arithmetic coder shall be terminated either at the end of all passes or at the end of every pass in a code-block. Figure C-4 shows two examples of termination patterns for the passes in a code-block. Which case is used is signaled in the COD or COC marker (see Annex A.6.1 and Annex A.6.2).

1 cleanup pass — **Arithmetic Coder (AC)**
2 significance propagation — **AC**
2 magnitude refinement pass — **AC**
2 cleanup pass — **AC**
• • •
final significance propagation — **AC**
final magnitude refinement pass — **AC**
final cleanup pass — **AC, terminate**

a) Termination only on last pass

1 cleanup pass — **AC, terminate**
2 significance propagation — **AC, terminate**
2 magnitude refinement pass — **AC, terminate**
2 cleanup pass — **AC, terminate**
• • •
final significance propagation — **AC, terminate**
final magnitude refinement pass — **AC, terminate**
final cleanup pass — **AC, terminate**

b) Termination on every pass

Figure C-4 — Examples of arithmetic coder termination patterns

When multiple terminations of the arithmetic coder are present, the length of each terminated segment is signalled in the packet header as described in Annex D.7.7.

C.5.1 Decoder termination

The decoder anticipates that the given number of codestream bytes will decode a given number of coding passes before the arithmetic coder is terminated. During decoding, bytes are pulled successively from the codestream until all the byte length for those coding passes has been reached. Often at that point there are more symbols to be decoded. Therefore, the decoder shall extend the input bit stream to the arithmetic coder with 0xFF bytes, as necessary, until all symbols have been decoded.

It is sufficient to append no more than two 0xFF bytes. This will cause the arithmetic coder to have at least one pair of consecutive 0xFF bytes at its input which is interpreted as an end-of-stream marker (see Annex B.3.4). The bit stream does not actually contain a terminating marker. However, the byte length is explicitly signalled enabling the terminating marker to be synthesized for the arithmetic coder.

NOTE — Two 0xFF bytes appended in this way is the simplest method, however, other equivalent extensions exist. This might be important since some arithmetic coder implementations might attach special meaning to the specific termination marker.

C.5.2 Arithmetic encoder termination

This termination is required if the predictable termination flag is on in the COD or COC markers (see Annex A.6.1 and Annex A.6.2). Otherwise, it is allowed, but not required.

It is important for the computation of fixed rate coding to be able to compute the number of bytes required to correctly decode all symbols up to any given truncation point, i.e. up to the end of the relevant coding passes. According to the termination style selected, a certain number of coding passes are performed before the arithmetic coder is terminated. The truncated length of the bit stream segment created must be estimated for rate control algorithms.

The FLUSH procedure performs this task adequately (see Annex B.2.9). However, since the FLUSH procedure increases the length of the codestream, and frequent termination may be desirable, other techniques may be employed. Any technique that places all of the needed bytes in the codestream in such a way that the decoder need not backtrack to find the position at which the next segment of the codestream should begin, is acceptable.

Using the notation of Annex B.2, the followings steps can be used:

1 Identify the number of bits in code register, C, which must be pushed out through the byte buffer. This is given by $k = (11 - CT_n) +1$

2 While (k > 0)

— Shift C left by CT and set CT = 0.

— Execute the BYTEOUT procedure. Note that this sets CT equal to the number of bits cleared out of the C register.

— Subtract CT from k.

3 Execute the BYTEOUT procedure to push the contents of the byte buffer register out to the codestream. Note that this step shall be skipped if the byte in the byte buffer has an 0xFF byte value.

The relevant truncation length in this case is simply the total number of bytes pushed out onto the codestream. The last byte output by the above procedure can generally be modified, within certain bounds, without affecting the symbols to be decoded. It will sometimes be possible to augment the last byte to the special value, 0xFF, which shall not be sent. It can be shown that this happens approximately 1/8 of the time.

C.6 Selective arithmetic coding bypass

This style of coding allows bypassing the arithmetic coder for the significance propagation and magnitude refinement passes in the fifth significant bit-plane, and the following bit-planes, of the block. The first cleanup pass (which is the first bit-plane of a block with a non-zero element) and the successive three significance propagation, magnitude refinement, and cleanup passes are coded with the arithmetic coder as before. The fourth cleanup pass includes an arithmetic coder termination (see Figure C-5).

Starting with the fourth significance propagation and magnitude refinement passes the bits that would have been returned from the arithmetic coder are instead returned after a routine that undoes the effects of bit stuffing. After each magnitude refinement pass the bit stream is “terminated” by padding to the byte boundary. The cleanup passes continue to receive data directly from the arithmetic coder and are always terminated.

The sign bit context is determined as in Annex C.4.2. However, the signbit is computed with Equation C.2, not Equation C.1.

$$signbit = AC(contextlabel) \qquad \text{C.2}$$

Whether this style is used is signaled in the COD or COC marker (see Annex A.6.1 and Annex A.6.2). Figure C-5 shows this progression.

1 cleanup pass — **Arithmetic Coding (AC)**
2 significance propagation — **AC**
2 magnitude refinement pass — **AC**
2 cleanup pass — **AC**
3 significance propagation — **AC**
3 magnitude refinement pass — **AC**
3 cleanup pass — **AC**
4 significance propagation — **AC**
4 magnitude refinement pass — **AC**
4 cleanup pass — **AC, terminate**
5 significance propagation — **raw**
5 magnitude refinement pass — **raw, terminate**
5 cleanup pass — **AC, terminate**

• • •

final significance propagation — **raw**
final magnitude refinement pass — **raw, terminate**
final cleanup pass — **AC, terminate**

Figure C-5 — Selective arithmetic coding bypass

The length of each terminated segment is signalled in the packet header as described in Annex D.7.7.

C.6.1 Undoing the effects of bit stuffing

If a 0xFF value is encountered in the bit stream, then the first bit of the next byte is discarded. The sequence of bits used in the selective arithmetic coding bypass have been stuffed into bytes using a bit stuffing routine.

At the encoder, bits are packed into bytes from the most significant bit to the least significant bit. Once a complete byte is assembled, it is emitted to the bit stream. If the value of the byte is an 0xFF a single zero bit is stuffed into the most significant bit of the next byte. Once all bits of the coding pass have been assembled, the last byte is packed to the byte boundary and emitted. The last byte shall not be an 0xFF value.

NOTE — Since the decoder appends 0xFF values, as necessary, to the bit stream representing the coding pass (see Annex C.5.1), truncation of the bit stream may be possible.

C.6.2 Predictable termination

This termination is required if the predictable termination flag is on in the COD or COC markers (see Annex A.6.1 and Annex A.6.2). Otherwise, it is allowed, but not required.

When all the bits from a the coding pass have been assembled by the encoder, the last byte is packed to the boundary with the sequence 0101010.

C.7 Vertically code-block scan causal context formation

This style of coding constrains the context formation to the current and past code-block scans (four rows of vertically scanned samples). That is, any coefficient from the next code-block scan are considered to be insignificant. Whether this style is used signaled in the COD or COC marker (see Annex A.6.1 and Annex A.6.2).

C.8 Flow diagram of the code-block coding

The steps for modeling each bit-plane of each code-block can be viewed graphically in Figure C-6. The decisions made are in Table C-7 and the bits and context sent to the coder are in Table C-8. These show the context model without the selective arithmetic coding bypass or the vertically causal model.

Table C-7 — Decisions in the context model flow chart

Decision	Question	Description
D0	Is this the first significance bit-plane for the code-block?	Annex C.4
D1	Is the current coefficient significant?	Annex C.4.1
D2	Is the context bin zero? (see Table C-1)	Annex C.4.1
D3	Did the current coefficient just become significant?	Annex C.4.1
D4	Are there more coefficients in the significance propagation?	
D5	Is the coefficient insignificant?	Annex C.4.3
D6	Was the coefficient coded in the last significance propagation?	Annex C.4.3
D7	Are there more coefficients in the magnitude refinement pass?	
D8	Are four contiguous undecoded coefficients in a column each with a 0 context?	Annex C.4.4
D9	Is the coefficient significant?	Annex C.4.4
D10	Are there more coefficients remaining of the four column coefficients?	
D11	Are the four contiguous bits all zero?	Annex C.4.4
D12	Are there more coefficients in the cleanup pass?	

Table C-8 — Coding in the context model flow chart

Code	Decoded symbol	Context	Brief explanation	Description
C0	_	—	Goto the next coefficient or column	
C1	Newly significant?	Table C-1, 9 context labels	Decode significant bit of current coefficient (significance propagation)	Annex C.4.1
C2	Sign bit	Table C-3, 5 context labels	Decode sign bit of current coefficient	Annex C.4.2
C3	Current magnitude bit	Table C-4, 3 context labels	Decode magnitude refinement pass bit of current coefficient	Annex C.4.3
C4	0 1	Run-length context label	Decode run-length of four zeros Decode run-length not of four zeros	Annex C.4.4

Table C-8 — Coding in the context model flow chart

Code	Decoded symbol	Context	Brief explanation	Description
C5	00 01 10 00	UNIFORM	First coefficient is first with non-zero bin Second coefficient is first with non-zero bin Third coefficient is first with non-zero bin Forth coefficient is first with non-zero bin	Annex C.4.4 and Table B-2

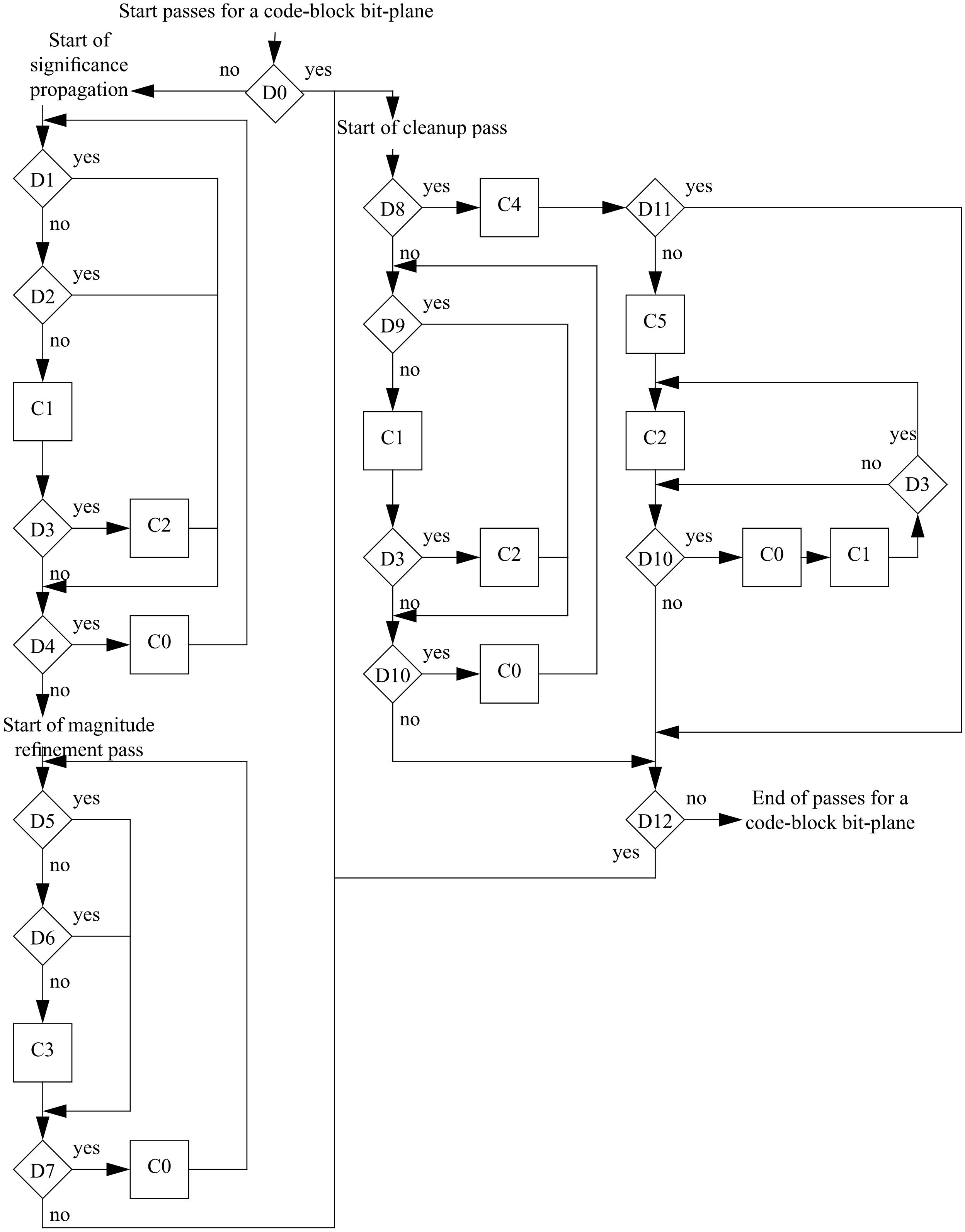


Figure C-6 — Flow chart for a all passes on a code-block bit-plane

Annex D

Bit stream ordering

(This annex forms an integral part of this Recommendation | International Standard)

This annex defines the bit stream ordering. It describes the image and image components, and the tiling of the image and tile-parts. It reviews the decomposition of the image tile-components with the wavelet transform into decomposition levels and sub-bands (see Annex F). It defines an order for the sub-bands.

The layering of the code-blocks for a decomposition level creating the data of a packet is described, as well as the packet header.

D.1 Image and components

The reference grid, image area offset, and component separation are defined by dimensional pairs (Xsiz, Ysiz), (XOsiz, YOsiz), and (XRsiz(i), YRsiz(i)) respectively in reference grid units. These are all from the SIZ marker (see Annex A.5.1). Thus, the reference grid is a rectangular grid of data points with the indices from (0,0) to (Xsiz-1, Ysiz-1).

The image area is a rectangle that is placed on the reference grid. The bottom right corner of the reference grid (Xsiz-1, Ysiz-1) corresponds to the bottom right corner of the image. The top left of the image area is offset from the origin of the reference grid by the image offset (XOsiz, YOsiz). Figure D-1 shows this correspondence. The size of the image is (Xsiz-XOsiz, Ysiz-YOsiz).

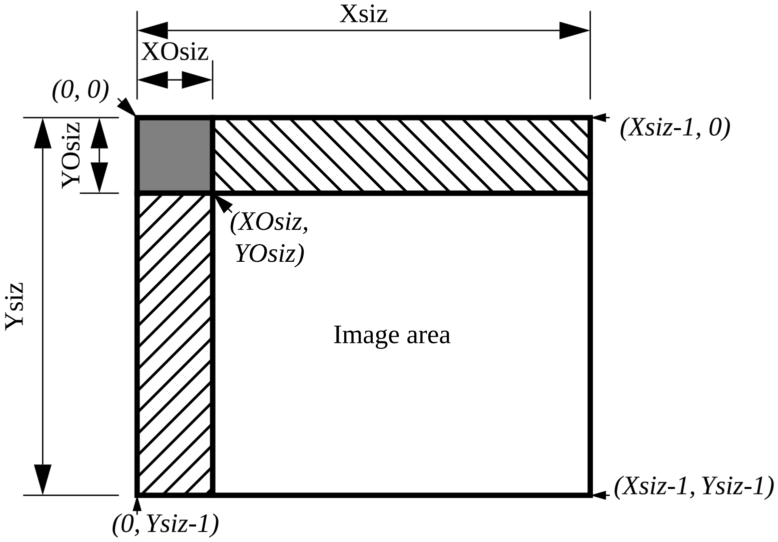
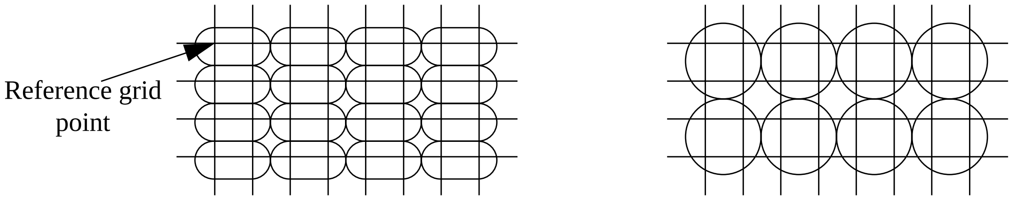


Figure D-1 — Reference grid diagram

Images are made up of components related to each other via the reference grid. The number of reference grid points separated by each component in both the X and Y directions is specified in the SIZ marker (XRsiz (component), YRsiz (component), see Annex A.5.1). This states that for a given component, i, for every XRsiz(i) reference grid points there are in the image area, there is one sample of component, i. Thus, the dimensions of component, i, are

$$\left(\left\lceil \frac{Xsiz}{XRsiz(i)} \right\rceil - \left\lceil \frac{XOsiz}{XRsiz(i)} \right\rceil, \left\lceil \frac{Ysiz}{YRsiz(i)} \right\rceil - \left\lceil \frac{YOsiz}{YRsiz(i)} \right\rceil\right) \qquad \text{D.1}$$

NOTE — Figure D-2 shows an example of how a component might be separated on the reference grid, given XRsiz and YRsiz. It is equivalent, for the purposes of this annex, to think of a particular grid point as the location of a components sample; e.g. the upper left. However, this might not be the correct location for proper colour representation.



a) component mapping to reference grid, XRsiz=2, YRsiz=1.

b) component mapping to reference grid, XRsiz=2, YRsiz=2.

Figure D-2 — Example of component separation

D.2 Image tiles

The reference grid is partitioned into a regular sized rectangular array of tiles. The tile size and tiling offset are defined by dimensional pairs (XTsiz, YTsiz), and (XTOsiz, YTOsiz) respectively in reference grid units. These are all from the SIZ marker (see Annex A.5.1).

Every tile is XTsiz reference grid points wide and YTsiz reference grid points high. The top left corner on the first tile (tile 0) is offset from the top left corner of the reference grid by (XTOsiz, YTOsiz). The tiles are numbered in raster order. That is from the top left to the top right, down one and to the left, etc. Thus, the first tile's starting coordinates are (XTOsiz, YTOsiz). Figure D-3 shows this relationship.

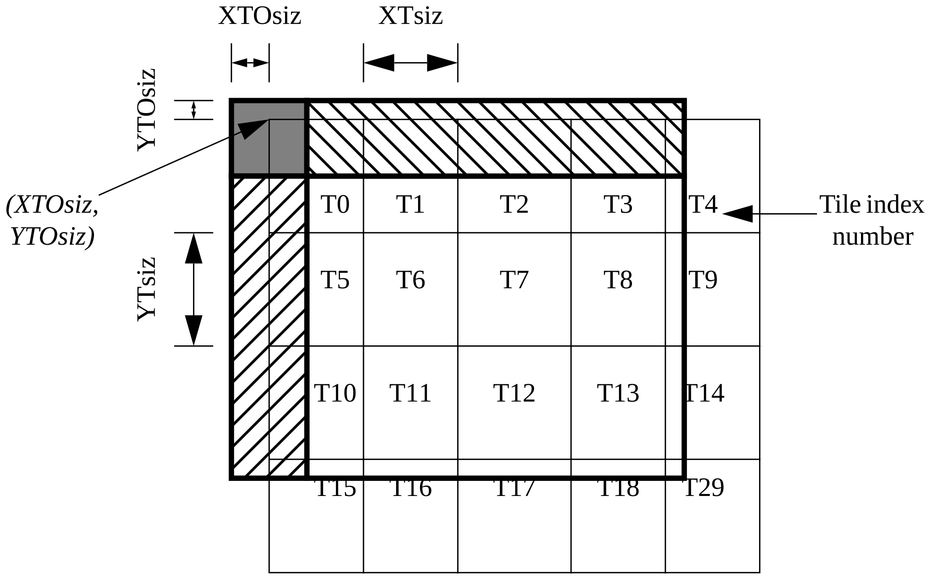


Figure D-3 — Tiling of the reference grid diagram

The tile offsets are constrained to be no greater than the image area offsets. This is expressed by the following ranges:

$$0 \le XTOsiz \le XOsiz \qquad 0 \le YTOsiz \le YOsiz \tag{D.2}$$

Also, the tile size plus the offset shall be greater than the image area offset. This insures that the first tile (tile 0) will contain at least one reference grid point from the image area. This is expressed by the following ranges:

$$XTsiz + XTOsiz > XOsiz \qquad YTsiz + YTOsiz > YOsiz \tag{D.3}$$

The number of tiles in the X direction (numXtiles) and the Ydirection (numYtiles) is the following:

$$numXtiles = \left\lceil \frac{Xsiz - XTOsiz}{XTsiz} \right\rceil \qquad numYtiles = \left\lceil \frac{Ysiz - YTOsiz}{YTsiz} \right\rceil. \tag{D.4}$$

For a given tile number, t, of any individual component, i, the number of samples horizontally, xnum (i, t), is the following:

$$\begin{aligned} xnum(i,t) = {} & min\left(\left\lceil \frac{Xsiz}{XRsiz(i)} \right\rceil, \left\lceil \frac{(mod(t, numXtiles) + 1) \cdot XTsiz + XTOsiz}{XRsiz(i)} \right\rceil\right) \\ & -max\left(\left\lceil \frac{XOsiz}{XRsiz(i)} \right\rceil, \left\lceil \frac{(mod((t, numXtiles))) \cdot XTsiz + XTOsiz}{XRsiz(i)} \right\rceil\right) \end{aligned} \tag{D.5}$$

Essentially the left side of the equation is all the samples up to the current right hand tile boundary. The right side of the equation is all the samples in previous tiles to the left. The maximum in the right hand side only comes into play in the first tile where the offset is subtracted from tile size. The minimum on the left hand side only comes into play when the last tile is less than the boundary of a tile.

Similarly, for a given tile number, *t*, of any individual component, *i,* the number of samples vertically, *ynum (i, t),* is the following:

$$\begin{aligned} ynum(i,t) = {} & min\left(\left\lceil \frac{Ysiz}{YRsiz(i)} \right\rceil, \left\lceil \frac{\left(\left\lfloor \frac{t}{numXtiles} \right\rfloor + 1\right) \cdot YTsiz + YTOsiz}{YRsiz(i)} \right\rceil\right) \\ & -max\left(\left\lceil \frac{YOsiz}{YRsiz(i)} \right\rceil, \left\lceil \frac{\left\lfloor \frac{t}{numXtiles} \right\rfloor \cdot YTsiz + YTOsiz}{YRsiz(i)} \right\rceil\right) \end{aligned} \tag{D.6}$$

The coordinates of a particular tile on the reference grid are described by the following equation:

$$tx_0(u, v) = max(XTOsiz + u \cdot XTsiz, XOsiz) \tag{D.7}$$

$$ty_0(u, v) = max(YOsiz + v \cdot YTsiz, YOsiz) \tag{D.8}$$

$$tx_1(u, v) = min(XTOsiz + u \cdot XTsiz, Xsiz) \tag{D.9}$$

$$ty_1(u, v) = min(YOsiz + v \cdot YTsiz, Ysiz) \tag{D.10}$$

where *(u, v)* denotes the horizontal and vertical indices (starting at 0) of the tile, *$tx_0(u,v)$* and *$ty_0(u,v)$* are the coordinates of the upper left corner of the tile, *$tx_1(u,v)-1$* and *$ty_1(u,v)-1$* are the coordinates of the lower right corner of the tile.

> NOTE — As an informative example, consider a case in which one would like to impose a tile structure with width-to-height ratio of 16:9 upon an image with actual width-to-height of 4:3. Specifically, suppose that the reference grid size of (Xsiz,Ysiz) = (1024,576), is large enough to accommodate the maximum width and maximum height of any image component. Suppose further that the component of interest is (640,480) in size.
>
> This component can span only one reference grid point per sample in each direction. Thus, (XRsiz, YRsiz) = (1,1) for this component. The size of the component is specified by setting the image area offset (XOsiz, YOsiz) to (384,96), so that, from Equation D.1, the component size is ((1024-384)/1, (576-96)/1) = (640,480).

Suppose that the tile size is chosen as (XTsiz,YTsiz) = (192,108). Imposing the tile grid at a reference grid offset of (XTOsiz, YTOsiz) = (224,12).

The number of tiles in each direction is given (from Equation D.4) by

$$(numXtiles, numYtiles) = \left(\left\lceil\frac{(1024-224)}{192}\right\rceil, \left\lceil\frac{(576-12)}{108}\right\rceil\right) = \left(\left\lceil\frac{800}{192}\right\rceil, \left\lceil\frac{564}{108}\right\rceil\right) = (5, 6)$$

The top row of tiles (0-4) will have height (from Equation D.6)

$$ynum = min\left(\left\lceil\frac{576}{1}\right\rceil, \left\lceil\frac{108+12}{1}\right\rceil\right) - max\left(\left\lceil\frac{96}{1}\right\rceil, \left\lceil\frac{12}{1}\right\rceil\right) = 120-96 = 24$$

The bottom row of tiles (25-29) will have height

$$ynum = min\left(\left\lceil\frac{576}{1}\right\rceil, \left\lceil\frac{(6\cdot 108)+12}{1}\right\rceil\right) - max\left(\left\lceil\frac{96}{1}\right\rceil, \left\lceil\frac{(5\cdot 108)+12}{1}\right\rceil\right) = 576-552 = 24$$

All other tiles will have height Ytsiz = 108.

The left column of tiles (0,5,10,15,20,25) will have width (from Equation D.5)

$$xnum = min\left(\left\lceil\frac{1024}{1}\right\rceil, \left\lceil\frac{192+224}{1}\right\rceil\right) - max\left(\left\lceil\frac{384}{1}\right\rceil, \left\lceil\frac{224}{1}\right\rceil\right) = 416-384 = 32$$

The right column of tiles (4,9,14,19,24,29) will have width

$$xnum = min\left(\left\lceil\frac{1024}{1}\right\rceil, \left\lceil\frac{(5\cdot 192)+224}{1}\right\rceil\right) - max\left(\left\lceil\frac{384}{1}\right\rceil, \left\lceil\frac{(4\cdot 192)+224}{1}\right\rceil\right) = 1024-992 = 32$$

All other tiles will have height Xtsiz = 192.

D.3 Tile-parts

While tiles are coherent geometric areas on the image, the coded data of a single tile may be broken into parts and distributed throughout the codestream. Each tile-part has at least one packet from that tile. Packets cannot be distributed across more than one tile-part. The tile number in the SOT marker (see Annex A.4.2) denotes to which tile the tile-part relates.

Tile-parts are ordered in such a way that the progression of the packets is preserved. Which packets are in the tile-parts is known dependent on the progression order for the tile (including progression order changes). That is, the first tile-part contains some number of the first packets. The second tile-part contains some number of packets immediately following those in the first tile-part, and so on. Tile-parts shall be decoded in order, as are packets. (Packets are described in Annex D.5.) The order in which tile-parts shall be decoded is signalled in the SOT marker (see Annex A.4.2).

The packet headers describe the contents of a tile-part (see Annex D.5). If a COD, COC, QCD, QCC, RIM, POM, POT or other marker for the entire tile is used, then it is present only in the first tile-part.

D.4 Wavelet decomposition level, resolution, sub-band, and code-block order

The sub-band order is defined in Annex F.2.9. The code-blocks divide the sub-bands of a tile as shown in Figure C-1 (see Annex C.1). Within each sub-band the code-blocks are visited in raster order. That is starting on the top left most block, proceeding across that row to the right most block, and then proceeding to the left most block of the next row, and so on.

Resolution is defined as the size of the image when certain decomposition levels have been decoded. As defined in Annex F.2.9, the lowest resolution, resolution 0 contains the packets of the nLL band, where n is the number of composition. Resolution 1 contains the packets from the nHL, nLH, and nHH sub-bands. Resolution 2 contains the packets for the (n-1)HL, (n-1)LH, and (n-1)HH sub-bands, and so on. There are n+1 resolutions.

D.5 Layering and packets

Layerings are groupings of bit-plane coding passes from blocks at the same decomposition level of the same component within a tile. Each layer is created by collecting some number of complete coding passes in order from each block. The actual number of coding passes for a given block can range from none to the entire block. The number of layers for each decomposition level is the same and is signaled in the COD marker (see Annex A.6.1).

Each layer successively and monotonically improves the image quality at the given resolution. Layers have an order in which the decoder shall be able to decode starting from the zeroth layer to the final layer contained in the bit-stream. Layers shall be decoded in this order.

For a given tile, the packets contain data from a specific layer, a specific resolution, and a specific component.

Packets represent information from a single tile within a single component, resolution, and layer. Within any given tile, component, and resolution the code-blocks contributing to each layer shall be represented by one, or more, packets in accordance with a packet partition.

D.5.1 Packet partition

Consider a particular tile, component, and resolution. The tile-component reconstructed from the sub-bands contained in the present resolution and the previous resolutions is described by the coordinates ax, ay, bx, by as shown in Figure D-4.

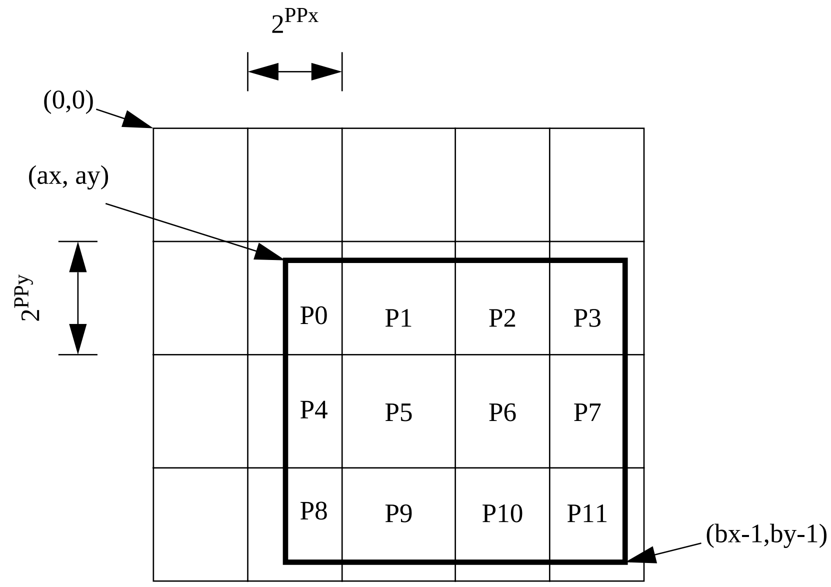


Figure D-4 — Packet partition

The coordinates are defined by the following

$$ax = \left\lceil \frac{tx_0(u, v)}{XRsiz(i) \cdot 2^{n-r}} \right\rceil \qquad ay = \left\lceil \frac{ty_0(u, v)}{YRsiz(i) \cdot 2^{n-r}} \right\rceil \qquad \text{D.11}$$

and

$$bx = \left\lceil \frac{tx_1(u, v)}{XRsiz(i) \cdot 2^{n-r}} \right\rceil \qquad by = \left\lceil \frac{ty_1(u, v)}{YRsiz(i) \cdot 2^{n-r}} \right\rceil \qquad \text{D.12}$$

where (tx,ty) are the coordinates of the upper left corner of current tile on the reference grid, r is the resolution, n is the number of decomposition levels for the tile-component, and i is the component.

The packet partition is described by the parameters *PPx* and *PPy* a shown in Figure D-4. The number of packets which span the tile-component at resolution, *r,* is given by

$$numpacketswide = \left\lceil \frac{bx}{2^{PPx}} \right\rceil - \left\lfloor \frac{ax}{2^{PPx}} \right\rfloor \qquad numpacketshigh = \left\lceil \frac{by}{2^{PPy}} \right\rceil - \left\lfloor \frac{ay}{2^{PPy}} \right\rfloor \tag{D.13}$$

The packet position index runs from 0 to numpackets - 1 where numpacketswide * numpacketshigh in raster order (see Figure D-4). This is used for packet ordering in the codestream.

The code-block partition in Annex C.1 shall respect the packet partition. Specifically, the code-block size for the sub-bands is determined as $2^{xcb'}$ by $2^{ycb'}$ where

$$xcb' = \begin{cases} min(xcb, PPx - 1), r > 0 \\ min(xcb, PPx), r = 0 \end{cases} \tag{D.14}$$

and

$$ycb' = \begin{cases} min(ycb, PPy - 1), r > 0 \\ min(ycb, PPy), r = 0 \end{cases} \tag{D.15}$$

It can happen that a packet is empty even when the tile-component contains only one packet. Empty packets shall be represented by a single byte according to Figure D.7.

NOTE — Packets can have data for only part of the layer. This occurs when the codestream has been truncated. However, if there is less than a complete layer in a packet, no successive layers can be in the codestream.

D.6 Progression order

For a given tile, the packets contain data from specific a layer, a specific resolution, the a specific component. The order in which these packets are interleaved is called the progression order. The interleaving of the packets can progress along four axes: partition position, layer, resolution, or component.

D.6.1 Progression order determination

Which of the five possible progression orders is designated in the COD or COC markers (see Annex A.6.1 and Annex A.6.2). These are described below

D.6.1.1 Layer progressive

Layer progression is defined as the interleaving of the packets in the following order:

```
for each l = 0, ... , L-1 where L is the number of layers,
    for each r = 0, ... , n where n is the number of decomposition levels,
        for each i = 0, ... , Csiz-1
            packets for component, i, resolution, r, and layer, l, appear in the order shown in Figure D-4.
```

D.6.1.2 Resolution-layer progressive

Resolution-layer progression is defined as the interleaving of the packets in the following order:

```
for each r = 0, ... , n where n is the number of decomposition levels,
    for each l = 0, ... , L-1 where L is the number of layers,
        for each i = 0, ... , Csiz-1
            packets for component, i, resolution, r, and layer, l, appear in the order shown in Figure D-4.
```

D.6.1.3 Resolution-position progressive

Resolution-position progression is defined as the interleaving of the packets in the following order:

```
for each r = 0, ... , n where n is the number of decomposition levels,
    for each y = $ty_0$, ... , $ty_1$-1,
        for each x = $tx_0$, ... , $tx_1$-1,
            for each i = 0, ... , Csiz-1
                if (y = $ty_0$ or y divisible by $YRsiz(i) \cdot 2^{PPy(r, i) + n - r}$)
                    if (x = $tx_0$ or x divisible by $XRsiz(i) \cdot 2^{PPx(r, i) + n - r}$)
                        for the next packet position in Figure D-4
                            for each l = 0, ... , L-1 where L is the number of layers
                                the packet at that position in component, i, resolution, r, and layer, l, appears.
```

To use this progression, XRsiz and YRsiz values must be powers of two for each component.

D.6.1.4 Position-component progressive

Position-component progression is defined as the interleaving of the packets in the following order:

```
for each y = $ty_0$, ... , $ty_1$-1,
    for each x = $tx_0$, ... , $tx_1$-1,
        for each i = 0, ... , Csiz-1
            for each r = 0, ... , n where n is the number of decomposition levels,
                if (y = $ty_0$ or y divisible by $YRsiz(i) \cdot 2^{PPy(r, i) + n - r}$)
                    if (x = $tx_0$ or x divisible by $XRsiz(i) \cdot 2^{PPx(r, i) + n - r}$)
                        for the next packet position in Figure D-4
                            for each l = 0, ... , L-1 where L is the number of layers
                                the packet at that position in component, i, resolution, r, and layer, l, appears.
```

To use this progression, XRsiz and YRsiz values must be powers of two for each component.

D.6.1.5 Component-position progressive

Resolution-position progression is defined as the interleaving of the packets in the following order:

```
for each i = 0, ... , Csiz-1
    for each y = $ty_0$, ... , $ty_1$-1,
        for each x = $tx_0$, ... , $tx_1$-1,
            for each r = 0, ... , n where n is the number of decomposition levels,
                if (y = $ty_0$ or y divisible by $YRsiz(i) \cdot 2^{PPy(r, i) + n - r}$)
                    if (x = $tx_0$ or x divisible by $XRsiz(i) \cdot 2^{PPx(r, i) + n - r}$)
                        for the next packet position in Figure D-4
                            for each l = 0, ... , L-1 where L is the number of layers
```

the packet at that position in component, i, resolution, r, and layer, l, appears.

D.6.1.6 Analogy in two dimensions

The progression defined above can be understood, by analogy, as a two progression as shown in Figure D-5.

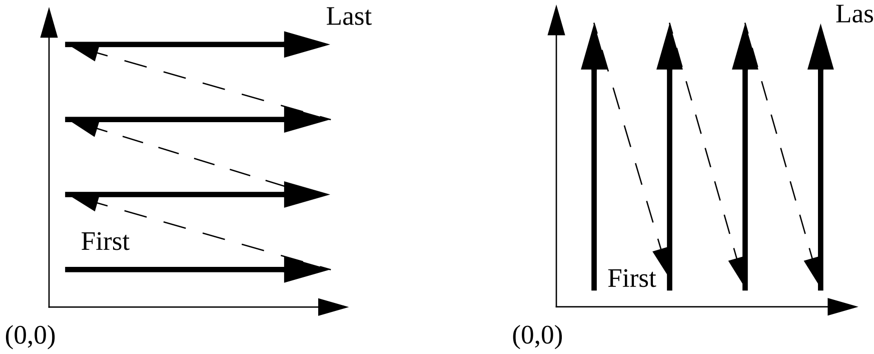


Figure D-5 — Examples of progression ordering in two dimensions

D.6.2 Progression order change

The progression order in a tile may be limited to a subset of the original space. The component, resolution, and layer indices are limited to the bounds defined in the POM or POT marker segments are present (see Annex A.6.6 and Annex A.6.7).

After completing the interleaving of this space, a new progression ordering can be designated. Such a bounding and progression change is present in a tile only if either the POM or POT marker segments are present (see Annex A.6.6 and Annex A.6.7). The coordinate indices for each successive bounding shall be equal to, or larger than, those of the previous bound.

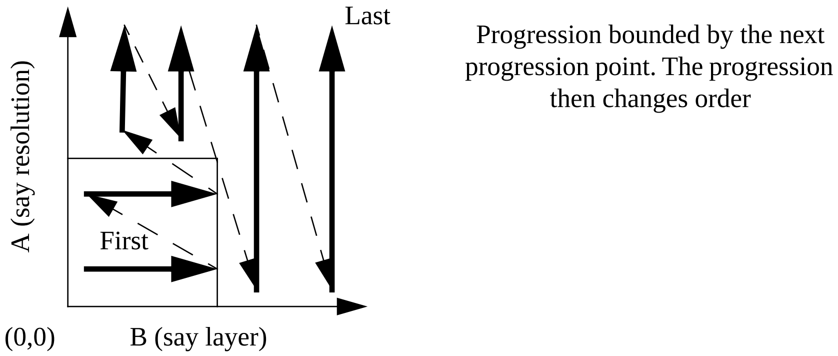

Last
Progression bounded by the next progression point. The progression then changes order
A (say resolution)
First
(0,0)
B (say layer)

Figure D-6 — Examples of progression ordering change in two dimensions

D.7 Packet header information coding

The packets have headers with the following information:

- Zero length packet
- Code-block inclusion
- Number of “insignificant” most significant bit-planes
- Number of coding passes for each code-block in this packet

— Length of the code-block data

This packet header is considered part of the bit stream and is integral to the proper decoding of the bit stream. Two items in the header are coded with a scheme called tag trees described below. The data bits of the packet header are packed into a whole number of bytes with a bit stuffing routine.

The packet headers are in the bit stream proceeding the packet data unless one of the following marker segments is used. If the PPM marker segment is used, all of the packet headers are removed from the bit stream and placed in the main header (see Annex A.7.4). If the PPM is not used, then a PPT marker segment may be used. In this case, all of the packet headers in that tile are removed from the bit stream and placed in the first tile-part header (see Annex A.7.5).

D.7.1 Bit stuffing routine

Bits are packed into bytes from the MSB to the LSB. Once a complete byte is assembled, it is emitted to the packet header in the bit stream. If the value of the byte is an 0xFF a single zero bit, not otherwise in the packet header, is stuffed into the MSB of the next byte. Once all bits of the packet header have been assembled, the last byte is packed to the byte boundary and emitted. The last byte shall not be an 0xFF value.

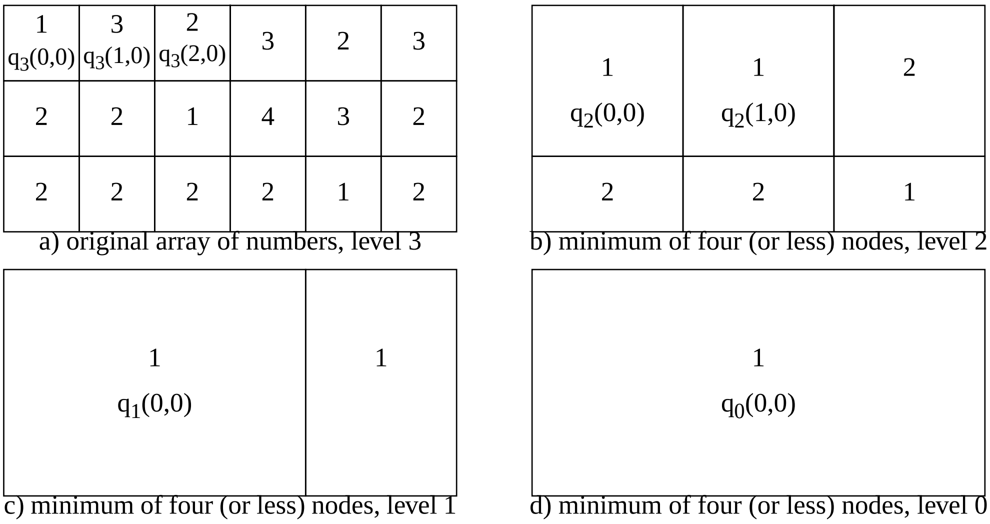
D.7.2 Tag trees

A tag tree is a way of representing a two dimensional array of whole numbers in a hierarchical way. It creates reduced resolution levels of the tree where at every node the minimum number of the (up to four) nodes below it is recorded. Figure D-7 shows an example of this representation.

The coding is the answer to a series of questions. Start with a value of zero. A 0 in the tag tree means that the minimum (or the number in the case of the highest level) is larger than the current value and a 1 means that the minimum (or the number in the case of the highest level) is equal the current value. For each contiguous 0 in the tag tree the current value is incremented by one. Nodes at higher levels cannot be coded until lower level nodes values are fixed (i.e a 1 is coded). The top node on level 0 is queried first. The next corresponding node on level 1 is then queried, and so on.

Only the information needed for the current code-block is stored at the current point in the packet header. The decoding of bits is ceased when sufficient information has been obtained. Also, the hierarchical nature of the tag trees means that the answers to many questions will have been answered when adjacent code-blocks and/or layers were coded. This information is not coded again. Therefore, there is a causality to the information in packet headers.

NOTE — For example, in the case above, the coding for the number at $q_3(0,0)$ would be 01111. The two bits, 01, imply that the top node at $q_0(0,0)$ is greater than zero and is, in fact one. The third bit, 1, implies that the node at $q_1(0,0)$ is also one. The fourth bit, 1, implies that the node at $q_2(0,0)$ is also one. And the final bit, 1, implies that the target node at $q_3(0,0)$ is also one. To decode the next node $q_3(1,0)$ the nodes at $q_0(0,0)$, $q_1(0,0)$, and $q_2(0,0)$ are already known. Thus, the bits coded are 001, the zero says that node at $q_3(1,0)$ is greater than 1, the second zero says it is greater than 2, and the one bit implies that the value is 3. Now that



a) original array of numbers, level 3

b) minimum of four (or less) nodes, level 2

c) minimum of four (or less) nodes, level 1

d) minimum of four (or less) nodes, level 0

Figure D-7 — Example of a tag tree representation

$q_3(0,0)$ and $q_3(1,0)$ are known, the code bits for $q_3(2,0)$ will be 101. The first 1 indicates $q_2(1,1)$ is one. The following 01 then indicates $q_3(2,0)$ is 2. This process continues for the entire array in Table D-7a.

D.7.3 Zero length packet

The first bit in the packet header denotes whether the packet is zero-length (not present). A zero value is zero length, a one value is non-zero length. A layer with zero length is a null layer. No code-blocks are included. No coefficients are coded or altered.

D.7.4 Code-block inclusion

In the sub-band order specified in Annex D.4, whether a block is included in this packet, or not, is coded. The code-blocks in a sub-band are tested for inclusion in raster order (top left to top right then second from top left to second from top right, etc.). For code-blocks that have not been included in a previous layer, this information is signaled with a separate tag tree code for each sub-band.

The values in the tag tree are the number of the layer in which the code-block is first included. Note that only the bits needed for determining whether the block is included are placed in the packet header. If some of the tag tree is already known from previous blocks or previous layers, it is not repeated. Likewise, only as much of the tag tree as is needed to determine inclusion in the current layer is included. If a code-block is not included until a later layer, then only a partial tag tree is included at that point in the bit stream.

For blocks that have been included in a previous layer, a 1 bit means that it is also included in this layer; a 0 means that it is not.

D.7.5 Number of all zero bit-planes before the first pass of a code-block

If a code-block is included for the first time, the number of bit-planes before a bit-plane that includes a coefficient with a one bit is coded. This number is coded with a separate independent tag tree for every sub-band, in the same manner as the code-block inclusion.

D.7.6 Number of coding passes

The number of coding passes included from each code-block is coded. The codewords used for this are shown in Table D-1.

Table D-1 — Codewords for the number of coding passes for each code-block

Number of coding passes	Codeword in Packet Header
1	0
2	10
3	1100
4	1101
5	1110
6 — 36	1111 0000 0 — 1111 1111 0
37 — 164	1111 11111 0000 000 — 1111 11111 1111 111

D.7.7 Length of the data from a given code-block

The packet header identifies the number of bytes contributed each included code-block. There are two cases: the code-block contains a single terminated bit stream from the arithmetic coder and the code-block contains multiple termination points.

D.7.7.1 Single termination point

The number of bits used to signal the number of bytes contributed by a code-block is Lblock plus $\log_2$ of the number of passes added in the code-block rounded down. Thus, layers with more passes are assumed to have more data. The value of Lblock is initial set to three. The number of bytes contributed by each block is proceeded by signaling bits that can increase value of Lblock, as needed. A signaling bit of zero indicates the current value of Lblock is sufficient. If there are k ones followed by a zero, the value of Lblock is incremented by k. While Lblock can only increase, the number of bits used to signal the code-block length can increase or decrease depending the number of coding passes included.

NOTE — For example, say that in successive layers a code-block has 6 bytes, 31 bytes, 44 bytes, and 134 bytes respectively, further assume that the number of coding passes is 1, 9, 2, and 5. The code for each would be 0 110 (0 delimits and 110=6), 0011111 (0 delimits, $\log_2 9 = 3$ bits for the 9 coding passes, 011111=31), 11 0 101100 (110 adds two bits. $\log_2 2 = 1$, 101100=44), and 1 0 10000110 (10 adds one bit, $\log_2 5 = 2$, 10000110=134).

NOTE — There is no requirement that the minimum number of bits be used to signal length (any number is valid).

D.7.7.2 Multiple termination points

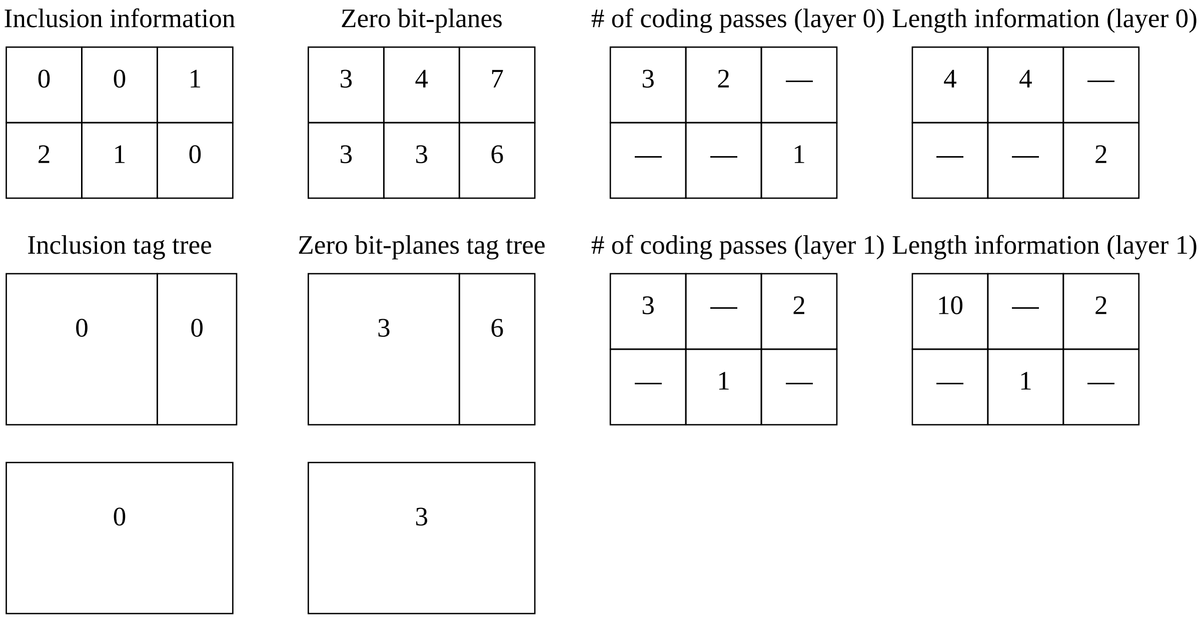
Let T be the set of terminated coding passes included for the code-block in the packet as indicated in Figure C-4 and Figure C-5. T is augmented with the final coding pass included in the packet. Let $n_1 < ... < n_K$ be an enumeration of T. K lengths are signaled consecutively with each length using the mechanism described in Annex D.7.7.1. The first length is the number of bytes from the start of the code-block's contribution to the packet to the end of coding pass n_1. The second length is the number of bytes from the end of coding pass, n_1, to the end of coding pass, n_2, and so on.

D.7.8 Order code-block packet header

The following is the packet header information order for one packet representing one layer of one decomposition level of one component in one tile-part.

```
bit for zero or non-zero length packet
    for each sub-band (LL first, then HL, LH, HH for n to 1)
for all code-blocks in raster order
    code-block inclusion bits (if not previously included then tag tree, else one bit)
    if code-block included
        if first instance of code-block
            zero bit-planes information
        number of coding passes included
        increase of code-block length indicator
        length of code-block contribution
```

Figure D-8 shows a brief example. This is the information known to the encoder. Table D-2 shows the resulting bit stream (in part) from this information.



Inclusion information

0	0	1
2	1	0

Zero bit-planes

3	4	7
3	3	6

# of coding passes (layer 0)

3	2	—
—	—	1

Length information (layer 0)

4	4	—
—	—	2

Inclusion tag tree

0	0

0

Zero bit-planes tag tree

3	6

3

# of coding passes (layer 1)

3	—	2
—	1	—

Length information (layer 1)

10	—	2
—	1	—

Figure D-8 — Example of the information known to the encoder

Table D-2 — Example packet header bit stream

Bit stream (in order)	Derived meaning
1	Packet non-zero in length
111	Block 0,0 included for the first time
000111	Block 0,0 insignificant for 3 bit-planes
1100	Block 0,0 has 3 coding passes included
0	Block 0,0 length indicator is unchanged
0100	Block 0,0 has 4 bytes 4 bits are used, 3 + floor($\log_2 3$)

Table D-2 — Example packet header bit stream

Bit stream (in order)	Derived meaning
1	Block 1,0 included for the first time
01	Block 1,0 insignificant for 4 bit-planes
10	Block 1,0 has 2 coding passes included
10	Block 1,0 length indicator is increased by 1 bit (3 to 4)
00100	Block 1,0 has 4 bytes 5 bits are used 4 + floor($\log_2 2$) (Note that while this is a legitimate entry, it is not minimal in code length.)
0	Block 2,0 not yet included
0	Block 0,1 not yet included
0	Block 1,1 not yet included
1	Block 2,1 included for the first time
00011	Block 2,1 insignificant for 6 bit-planes
0	Block 2,1 has 1 coding passes included
0	Block 2,1 length information is unchanged (3 bits)
010	Block 2,1 has 2 bytes 3 + $\log_2 1$ bits used
•••	Packet header data for the other sub-bands, coded packet data

Packet for the next layer

1	Packet non-zero in length
1	Block 0,0 included again
1100	Block 0,0 has 3 coding passes included
0	Block 0,0 length information is unchanged
1010	Block 0,0 has 10 bytes 3 + $\log_2$ (3) bits used
0	Block 1,0 not included in this layer
1	Block 2,0 included for the first time
01	Block 2,0 insignificant for 7 bit-planes
10	Block 2,0 has 2 coding passes included

Table D-2 — Example packet header bit stream

Bit stream (in order)	Derived meaning
0	Block 2,0 length information is unchanged
0010	Block 2,0 has 2 bytes $3 + \log_2 2$ bits used
0	Block 0,1 not yet included
1	Block 1,1 included for the first time
1	Block 1,1 insignificant for 3 bit-planes
0	Block 1,1 has 1 coding passes included
0	Block 1,1 length information is unchanged
001	Block 1,1 has 1 byte
0	Block 2,1 not included in this layer
•••	Packet header data for the other sub-bands, packet data

Annex E

Quantization

(This annex forms an integral part of this Recommendation | International Standard)

This Annex specifies the forms of quantization and dequantization used for encoding and reconstruction of image tile components. Quantization is the process by which the coefficients are reduced in precision. This operation is lossy unless the quantization step is one and the coefficients are integer, as produced by the reversible integer 5-3 wavelet.

E.1 Scalar coefficient quantization (informative)

After the Forward Wavelet Transform (see Annex F), each of the transform coefficients $a_b(u, v)$ of the sub-band b is quantized to the value $q_b(u, v)$ according to the following equation:

$$q_b(u, v) = sign(a_b(u, v)) \cdot \left\lfloor \frac{|a_b(u, v)|}{\Delta_b} \right\rfloor \tag{E.1}$$

The quantization step size Δ_b is represented relative to the dynamic range R_b of sub-band b , by the exponent ε_b and mantissa μ_b as:

$$\Delta_b = 2^{R_b - \varepsilon_b}\left(1 + \frac{\mu_b}{2^{11}}\right) \tag{E.2}$$

The dynamic range R_b depends on the number of bits used to represent the original image tile component and on the choice of the wavelet transform. The explicit determination of R_b is presented in Section E.2.

In order to prevent possible overflow or excursion beyond the nominal range of the integer representation of $|q_b(u, v)|$ arising, for example during floating point calculations, the number of bits for the integer representation really used at the encoder side is:

$$M_b = G + \varepsilon_b - 1 \tag{E.3}$$

where the number G of additional bits, called guard bits, has to be specified in the QCD or QCC marker (see Annex A.6.4 and Annex A.6.5). If a ROI is defined, then the number of magnitude bit is modified accordingly (see Annex H).

NOTE — Typical values for the number of guard bits are G =1 or G =2.

For reversible compression, the quantization step size is required to be 1. This implies that $\mu_b = 0$ and $R_b = \varepsilon_b$. In this case, only the exponent ε_b has to be recorded in the bit stream in the QCD or QCC markers (see Annex A.6.4 and Annex A.6.5). For irreversible compression, no particular selection of the quantization step size is required in this Specification and different applications may specify the quantization step sizes according to specific image tile component characteristics. However, one effective way of selecting the quantizer step size for one sub-band b is to normalize a default step size Δ_d with respect to the vertical and horizontal synthesis filters which are used in that specific sub-band. As an example, a typical choice for the quantization step size for sub-band $b = 2LH$ is:

$$\Delta_b = \frac{\Delta_d \cdot 2^{R_b}}{\sqrt{\left(\sum_{k \in Z} (l_p \otimes l_p)^2(k)\right) \times \left(\sum_{k \in Z} (l_p \otimes h_p)^2(k)\right)}} \tag{E.4}$$

with

$$(l_p \otimes h_p)(i) = \sum_{k \in Z} l_p(i) \cdot h_p(i-k) \qquad \text{E.5}$$

and where l_p and h_p are the impulse response of the low and high pass synthesis 1D filters (see Table E.1) used during the inverse filtering, and Δ_d is a default step size. A typical value for Δ_d is 2^{1-R_I} where R_I is the bit depth of the original tile image component.

Table E-1 — Impulse response of the low and high pass synthesis filter for the 9-7 wavelet transform

i	$l_p(i)$	$h_p(i)$
0	1.115087052456994	0.6029490182363579
±1	0.5912717631142470	-0.2668641184428723
±2	-0.05754352622849957	-0.07822326652898785
±3	-0.09127176311424948	0.01686411844287495
±4	0	0.02674875741080976
all other values	0	0

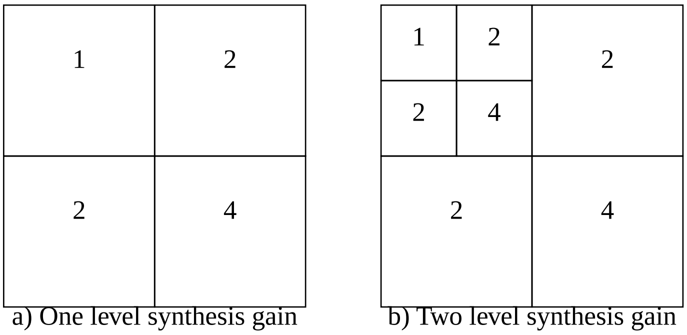
E.2 Scalar coefficient dequantization (normative)

At the decoder, the quantization step-size for each sub-band are retrieved from the bit stream using Equations E.2 where ε_b and μ_b are derived from the SPqcd^i parameters defined in the QCD or QCC markers (see Annex A.6.4 and Annex A.6.5), and R_b is the sum of the number of bits used to represent the original image tile component specified by the SIZ marker (see Annex A.5.1) and the base 2 exponent of the synthesis gain of the current sub-band. If a colour space transform, such as the RCT, is used, the number of bits used to represent the original image tile component is modified accordingly (see Annex G). For the 9-7 wavelet transform, all synthesis gains are equal to 1 (i.e. the corresponding base 2 exponent is equal to 0). For the 5-3 wavelet transform, the synthesis gain of a sub-band is recursively defined as the synthesis gain of the previous sub-band multiplied by the respective gains of the horizontal and vertical filters used to produced that sub-band. The low-pass filter has a unit gain, while the high-pass filter has a gain of 2. Figure E-1 shows the synthesis gain of each sub-band for one and two levels of the 5-3 wavelet transform decomposition and Figure E-2 presents the corresponding dynamic range R_b for each sub-band.

The exponent/mantissa pairs (ε_b, μ_b) are either explicitly signaled in the bit stream syntax for every sub-band, this is referred to as explicit quantization, or only signaled in the bit stream for the LL band (see Table A-27). In the latter case, known as implicit quantization, all other exponent/mantissa pairs (ε_b, μ_b) are derived implicitly from the single exponent/ mantissa pair (ε_o, μ_o) corresponding to the LL band, according to:

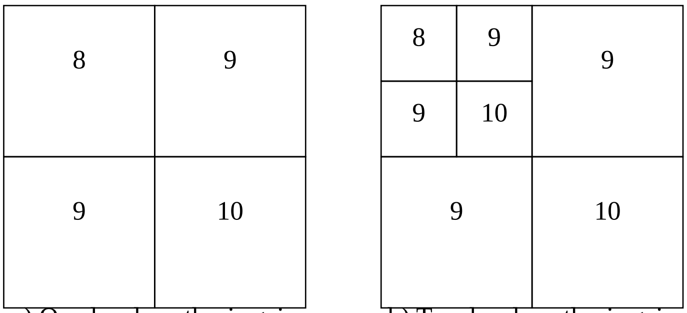
$$(\varepsilon_b, \mu_b) = (\varepsilon_o + nsd_b - nsd_o, \mu_o) \qquad \text{E.6}$$

where nsd_b denotes the number of sub-band decomposition levels from the original image tile component to the sub-band b.



a) One level synthesis gain b) Two level synthesis gain

Figure E-1 — Synthesis gain of each sub-band of the 5-3 wavelet transform decomposition



It is assumed in this example that no region of interest is defined in the image tile component and that the image tile component is defined with an 8 bit accuracy

a) One level synthesis gain b) Two level synthesis gain

Figure E-2 — Dynamic range R_b for each sub-band of the 5-3 wavelet transform decomposition

The maximum number M_b of encoded bit-planes (see Annex C) which can be expected in the code stream for sub-band b is retrieved by using Equation E.3 where the number of guard bits G is specified in the QCD or QCC markers (see Annex A.6.4 and Annex A.6.5). Although the encoder might have encoded all the bit-planes of all sample in sub-band b, due to the embedded nature of the code stream, a decoder may decide to decode only N_b bit-planes for a particular coding-block of the sub-band b. This is equivalent as to the use of a scalar quantizer with step size $2^{M_b - N_b} \cdot \Delta_b$ for all the sample of this coding-block. Due to the nature of the three coding passes (see Annex C), the step-size used in practice when truncation of the bit stream occurs may be different for different samples within the same coding-blocks if one bit-plane is not completely decoded. However, these step-sizes are always multiples of the reference step size by some power of two. Each decoded coefficient $\bar{q}_b(u, v)$ of sub-band b is used to reconstruct a transform coefficient $Rq_b(u, v)$ using for the 9-7 float wavelet transform the following equation

$$Rq_b(u, v) = \begin{cases} (\overline{q_b}(u, v) + r2^{M_b - N_b(u, v)}) \cdot \Delta_b & for\ \overline{q_b}(u, v) > 0 \\ (\overline{q_b}(u, v) - r2^{M_b - N_b(u, v)}) \cdot \Delta_b & for\ \overline{q_b}(u, v) < 0 \\ 0 & for\ \overline{q_b}(u, v) = 0 \end{cases} \qquad \text{E.7}$$

where $N_b(u, v)$ is the number of decoded bit-plane for sample $\bar{q}_b(u, v)$.

NOTE — The value r is the coefficient reconstruction value and is in the range of $0 \le r < 1$. It may be chosen to produce the best visual or objective quality for reconstruction. A typical value is r=1/2.

In the case of the integer 5-3 integer wavelet transform, the reconstructed transform coefficient $Rq_b(u, v)$ is recovered differently depending whether the bit stream has been truncated or not. If the bit stream is completely decoded (no truncation occurs) then $Rq_b(u, v) = \overline{q_b}(u, v)$ otherwise to reconstruct a transform coefficient $Rq_b(u, v)$, the following formula is used:

$$Rq_b(u,v) = \begin{cases} \left\lfloor (\overline{q_b}(u,v) + r2^{M_b - N_b(u,v)}) \cdot \Delta_b \right\rfloor & for\ \overline{q_b}(u,v) > 0 \\ \left\lfloor (\overline{q_b}(u,v) - r2^{M_b - N_b(u,v)}) \cdot \Delta_b \right\rfloor & for\ \overline{q_b}(u,v) < 0 \\ 0 & for\ \overline{q_b}(u,v) = 0 \end{cases} \qquad \text{E.8}$$

Annex F

Transformation of image tile components

(This annex forms an integral part of this Recommendation | International Standard)

This Recommendation | International Standard describes the forward transformation applied to one image tile component and specifies the inverse transformation used to reconstruct the image tile component.

F.1 Introduction and overview

Consider the tile defined by the coordinates tx_0, tx_1, ty_0 and ty_1 given in Equation D.7, Equation D.8, Equation D.9 and Equation D.10 in Annex D. Then the sample values $I(x, y)$ of image tile component c lie in the range defined by:

$$xtcpos \le x < xtcpos + xtcsiz \quad \text{and} \quad ytcpos \le y < ytcpos + ytcsiz \tag{F.1}$$

where

$$xtcpos = \left\lceil \frac{tx_0}{XRsiz(c)} \right\rceil \text{ and } xtcsiz = \left\lceil \frac{tx_1}{XRsiz(c)} \right\rceil - xtcpos \ , \tag{F.2}$$

$$ytcpos = \left\lceil \frac{ty_0}{YRsiz(c)} \right\rceil \text{ and } ytcsiz = \left\lceil \frac{ty_1}{YRsiz(c)} \right\rceil - ytcpos \tag{F.3}$$

F.1.1 Forward Transformation

As illustrated in Figure F-1, the forward transformation transforms image tile component samples $I(x, y)$ into sub-band samples $a_b(u, v)$. First, all the image component samples $I(x, y)$ of tile $T_{t,c}$ are DC level shifted into samples $I'(x,y)$. They are then transformed by the Forward Discrete Wavelet Transform (FDWT) to produce transform samples $J(u, v)$. The transform samples $J(u, v)$ are finally rearranged into sub-bands $a_b(u, v)$.

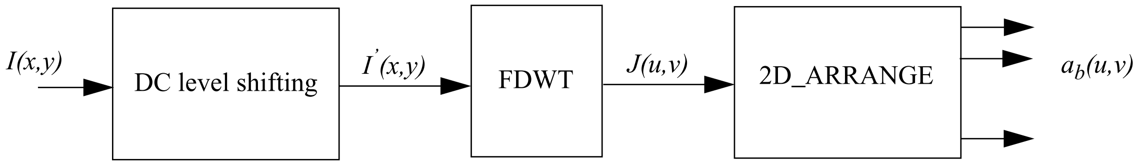


Figure F-1 — Forward Transformation

F.1.2 Inverse Transformation

As illustrated in Figure F-2, the inverse transformation transforms sub-band samples $a_b(u, v)$ into image tile component samples $I(x, y)$. First, sub-band coefficients $a_b(u, v)$ are rearranged into transform samples $J(u, v)$. They are then inverse transformed by the Inverse Discrete Wavelet Transform (IDWT) to produce level-shifted image tile component samples $I'(x,y)$. Finally all the image tile component samples $I'(x,y)$ of tile are inverse-level shifted.

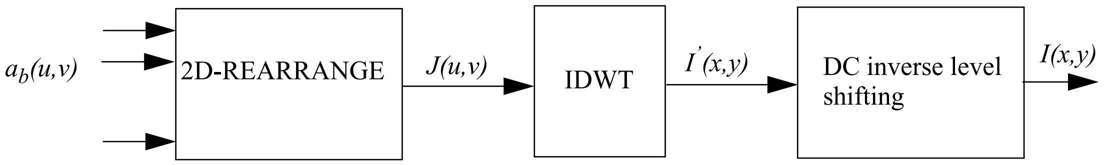


Figure F-2 — Inverse Transformation

F.1.3 One-dimensional sub-band decomposition

To perform the FDWT, this Recommendation | International Standard uses a one-dimensional sub-band decomposition of a one-dimensional set of samples into low-pass samples, representing a downsampled low-resolution version of the original set, and high-pass samples, representing a downsampled residual version of the original set, needed to perfectly reconstruct the original set from the low-pass set.

To perform the IDWT, this Recommendation | International Standard uses a one-dimensional sub-band recomposition of a one-dimensional set of samples from low-pass and high-pass samples.

F.1.4 Reversible and irreversible transformations

This Recommendation | International Standard uses one reversible transformation and one irreversible transformation. Given that image tile component samples are integer-valued, a reversible transformation requires the specification of a rounding procedure for intermediate non-integer-valued transform coefficients.

F.2 Forward Transformation (informative)

This Recommendation | International Standard uses FDWTs to transform image tile components.

F.2.1 DC level shifting of image tile components

Prior to computation of the FDWT, if the MSB of Ssiz is zero, then all samples $I(x, y)$ of the image tile component are level shifted by subtracting the same quantity from each sample:

$$I'(x, y) = I(x, y) - (2^{Ssiz^c - 1}) \qquad \text{F.4}$$

where $Ssiz^c$ is the component's precision parameter specified in Annex A.

F.2.2 The FDWT procedure

As illustrated in Figure F-3, the FDWT takes as input the level shifted image tile component samples $I'(x,y)$ and produces as output the transform image tile component samples $J(u,v)$, where N_L is a parameter of the FDWT representing a number of iterations, known as the number of levels of decomposition. The number of decomposition levels N_L are signalled in the COD or COC markers (see Annex A.6.1 and Annex A.6.2). The number of transform samples $J(u,v)$ is the same as the number of level-shifted image tile component samples $I'(x,y)$. The coordinates (x,y) and (u,v) span the same range.

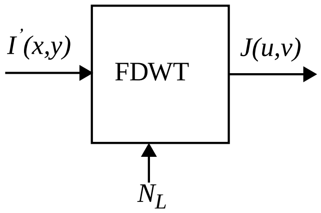


Figure F-3 — Parameters of the FDWT procedure

As illustrated in Figure F-4, the FDWT procedure starts with the initialization of the variable *lev* to zero, and the initialization of the samples *J(u,v)* to the values of $I'(x,y)$. The 2D_SD procedure is performed at level *lev*, where the level *lev* increases at each iteration, and until N_L iterations are performed.

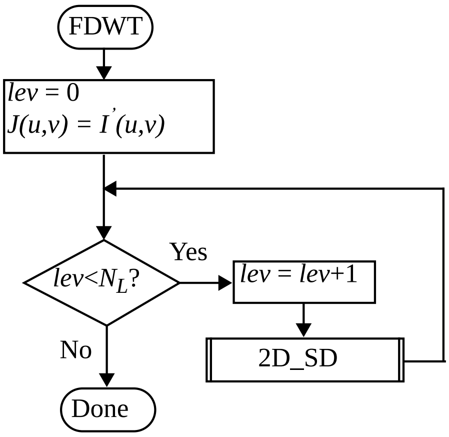


Figure F-4 — The FDWT Procedure

F.2.3 The 2D_SD procedure

As illustrated in Figure F-5, the 2D_SD procedure modifies a portion of the samples *J(u,v)*, according to the value of the variable *lev*, i.e. the current level of decomposition.

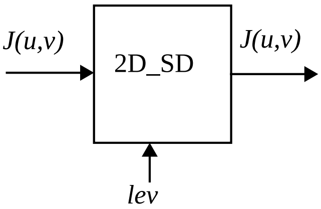


Figure F-5 — Parameters of the 2D_SD procedure

As illustrated in Figure F-6, it begins by applying the VER_SD procedure to all columns of *J(u,v)* for which *u* is a multiple of $2^{lev\text{-}1}$. It then applies the HOR_SD procedure to all rows of *J(u,v)* for which *y* is a multiple of $2^{lev\text{-}1}$.

F.2.4 The VER_SD procedure

As illustrated in Figure F-7, the VER_SD procedure modifies the sample values of column *u* of *J(u,v)*, for which *v* is a multiple of $2^{lev\text{-}1}$.

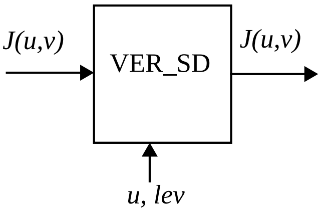
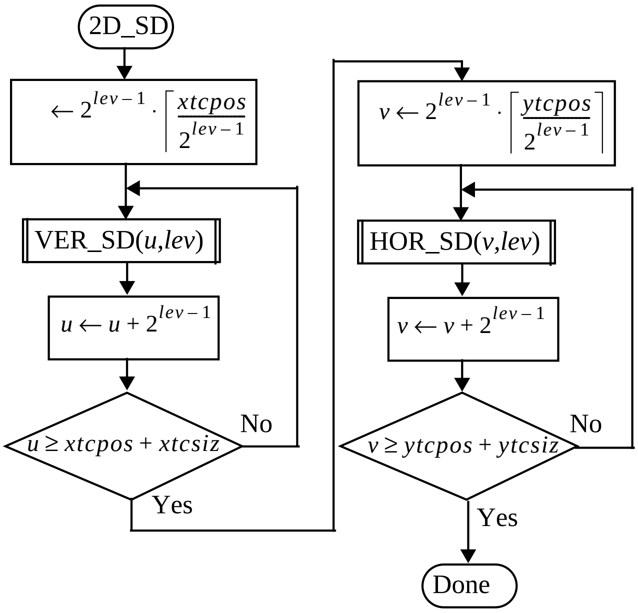


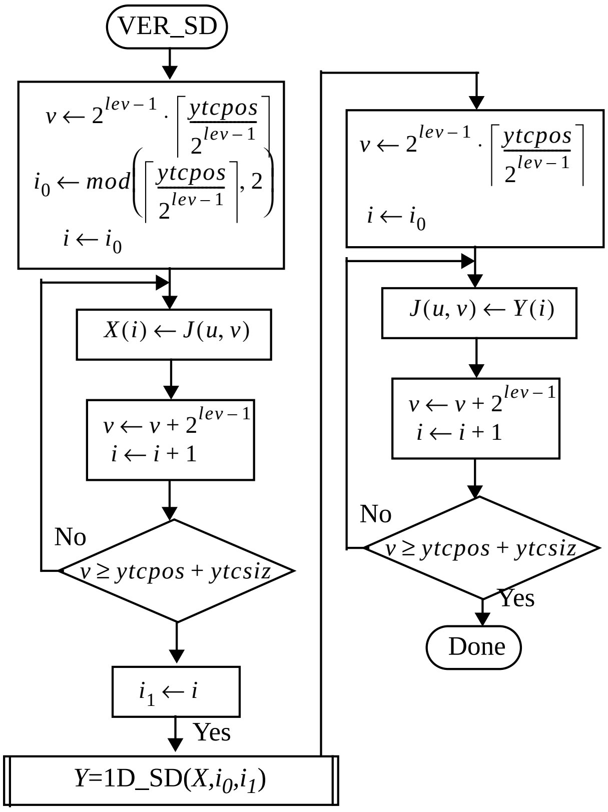
Figure F-7 — Parameters of the VER_SD procedure



2D_SD
$\leftarrow 2^{lev-1} \cdot \left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil$
VER_SD(u,lev)
$u \leftarrow u + 2^{lev-1}$
$u \geq xtcpos + xtcsiz$
No
Yes
$v \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
HOR_SD(v,lev)
$v \leftarrow v + 2^{lev-1}$
$v \geq ytcpos + ytcsiz$
No
Yes
Done

Figure F-6 — The 2D_SD procedure

As illustrated in Figure F-8, the VER_SD procedure stores all the sample values of column *u* of *J(u,v)*, for which *v* is a multiple of 2^{lev-1} into an array *X*, applies the 1D_SD procedure to array *X* and produces array *Y*. Then the values of array *Y* are replaced into the sample values of column *u* of *J(u,v)*, for which v is a multiple of 2^{lev-1}.



VER_SD
$v \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
$i_0 \leftarrow mod\left(\left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil, 2\right)$
$i \leftarrow i_0$
$X(i) \leftarrow J(u, v)$
$v \leftarrow v + 2^{lev-1}$
$i \leftarrow i + 1$
$v \geq ytcpos + ytcsiz$
No
$i_1 \leftarrow i$
Yes
Y=1D_SD(X,i0,i1)
$v \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
$i \leftarrow i_0$
$J(u, v) \leftarrow Y(i)$
$v \leftarrow v + 2^{lev-1}$
$i \leftarrow i + 1$
No
$v \geq ytcpos + ytcsiz$
Yes
Done

Figure F-8 — The VER_SD procedure

F.2.5 The HOR_SD procedure

As illustrated in Figure F-9, the HOR_SD procedure modifies the sample values of row *v* of *J(u,v),* for which *u* is a multiple of $2^{lev\text{-}1}$.

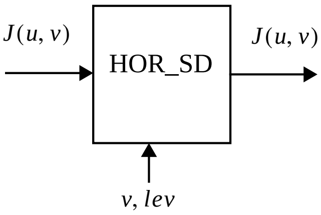


Figure F-9 — Parameters of the HOR_SD procedure

As illustrated in Figure F-10, the HOR_SD procedure stores all the sample values of row *v* of $J_{t,c}(u,v)$, for which *u* is a multiple of $2^{lev\text{-}1}$ into an array *X*, applies the 1D_SD procedure to array *X* and produces array *Y.* Then the values of array *Y* are replaced into the sample values of row *v* of *J(x,y),* for which *u* is a multiple of $2^{lev\text{-}1}$.

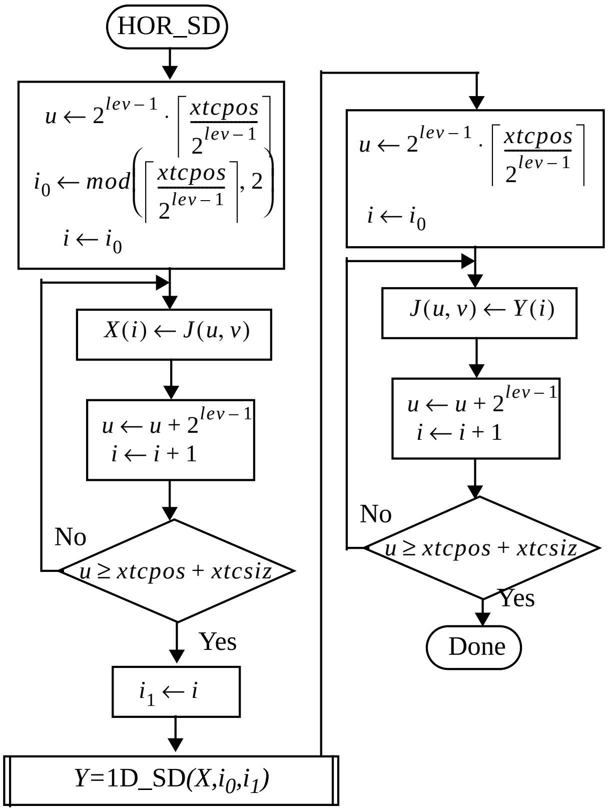

HOR_SD
$u \leftarrow 2^{lev-1} \cdot \left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil$
$i_0 \leftarrow mod\left(\left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil, 2\right)$
$i \leftarrow i_0$
$X(i) \leftarrow J(u, v)$
$u \leftarrow u + 2^{lev-1}$
$i \leftarrow i + 1$
No
$u \geq xtcpos + xtcsiz$
Yes
$i_1 \leftarrow i$
Y=1D_SD(X,i_0,i_1)
$u \leftarrow 2^{lev-1} \cdot \left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil$
$i \leftarrow i_0$
$J(u, v) \leftarrow Y(i)$
$u \leftarrow u + 2^{lev-1}$
$i \leftarrow i + 1$
No
$u \geq xtcpos + xtcsiz$
Yes
Done

Figure F-10 — The HOR_SD procedure

F.2.6 The 1D_SD procedure

As illustrated in Figure F-11, the 1D_SD procedure takes as input an array *X*, the index i_0 (which has value of 0 or 1) to the first sample of array *X*, an index i_1 to the sample following the last sample value of array *X*. Both i_0 and i_1 are calculated by the VER_SD or HOR_SD procedure which is calling the 1D_SD procedure.

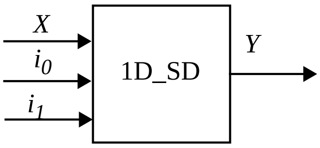


Figure F-11 — Parameters of the 1D_SD procedure

As illustrated in Figure F-12, the 1D_SD procedure first uses the 1D_EXT procedure to extend the signal *X* beyond its left and right boundaries resulting in the extended signal X_{ext}, and then uses the 1D_FILT procedure to filter the extended signal X_{ext} and produce the desired filtered signal *Y*.

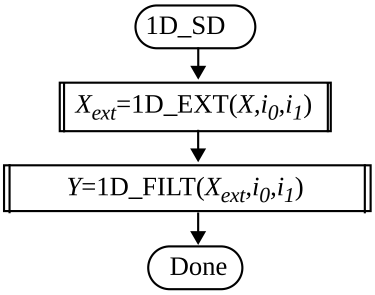


Figure F-12 — The 1D_SD procedure

F.2.7 The 1D_EXT procedure

As illustrated in Figure F-13 and Figure F-14, the 1D_EXT procedure extends signal *X* by i_{left} samples to the left and i_{right} samples to the right. The extension of the signal is required to enable filtering at both boundaries of the signal.

The first sample of signal *X* is sample i_0, and the last sample of signal *X* is sample i_1-1. This extension procedure is known as “periodic symmetric extension”. Symmetric extension consists in extending the signal with the signal samples obtained by a reflection of the signal centered on the first sample (sample i_0) for extension to the left, and in extending the signal with the signal samples obtained by a reflection of the signal centered on the last sample (sample i_1-1) for extension to the right. Periodic symmetric extension applies to the case where the number of samples by which to extend the signal on any one side exceeds the signal length: this case may happen at lower levels of decomposition. The procedure is one among possibly others which implements periodic symmetric extension.

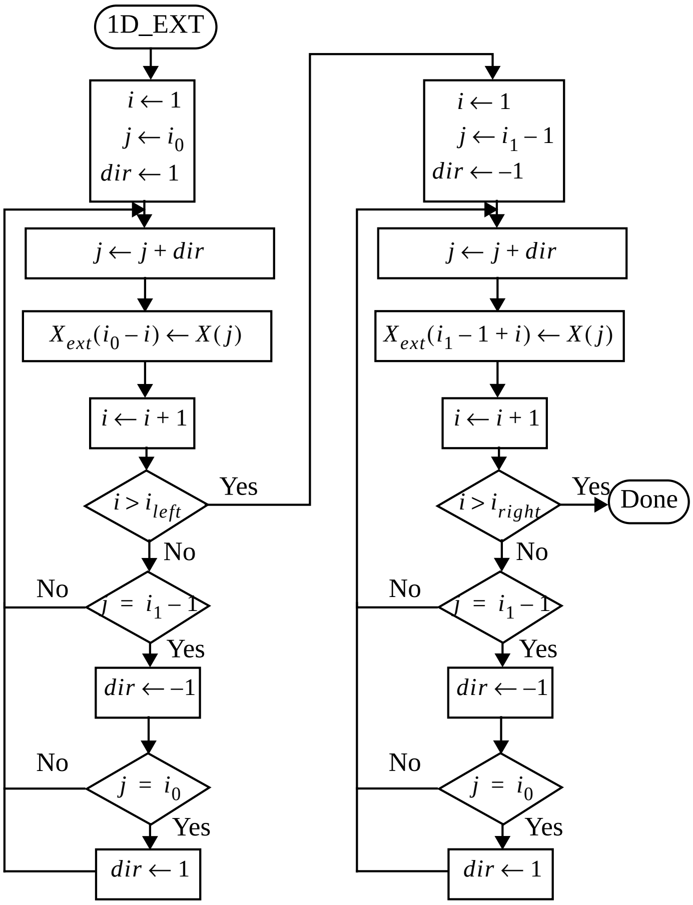


Figure F-13 — 1D_EXT procedure implementing periodic symmetric extension

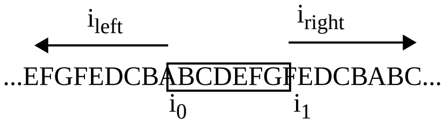


Figure F-14 — Periodic symmetric extension of signal

F.2.8 The 1D_FILT procedure

This Recommendation | International Standard uses exclusively one irreversible or one reversible transformation of image tile components. The transformation is reversible if the 1D-FILT procedure is reversible. The transformation is irreversible if the 1D_FILT procedure is irreversible. One irreversible procedure 1D_FILT$_I$ and one reversible filtering procedure 1D_FILT$_R$ is described.

As illustrated in Figure F-15, both procedures take as input an extended 1D signal X_{ext}, the index of the first sample i_0, and the index of the sample i_1 immediately following the last sample (i_1-1) They both produce the output signal Y. The even samples of the *Y* signal are a low-pass downsampled version of the extended signal X_{ext}, while the odd samples of the signal *Y* are a high-pass downsampled version of the extended signal X_{ext}.

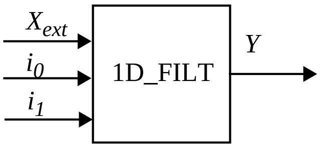


Figure F-15 — Parameters of the 1D_FILT procedure

Both the irreversible and reversible transformations are described using lifting-based filtering [15], which is a very efficient implementation of the DWT. Lifting-based filtering consists of a sequence of very simple filtering operations for which alternately, odd sample values of the signal are updated with a weighted sum of even sample values, and even sample values are updated with a weighted sum of odd sample values.

F.2.8.1 Irreversible 1D filtering

The irreversible transformation described in this section is the lifting-based DWT implementation of filtering by the Daubechies 9/7 filter [4].

Equation F.5 describes the 4 "lifting" steps (1 through 4) and the 2 "scaling" steps (5 and 6) of the 1D filtering performed on the extended signal $X_{ext}(n)$ to produce the i_1 samples of signal *Y*.

Step 1 is performed for all values of *n* such that $i_0 - 3 \le 2n + 1 < i_1 + 3$. Step 2 is then performed for all values of *n* such that $i_0 - 2 \le 2n < i_1 + 2$. Step 3 is then performed for all values of *n* such that $i_0 - 1 \le 2n + 1 < i_1 + 1$. Step 4 is performed for all values of *n* such that $i_0 \le 2n < i_1$. Each of these steps is performed on the entire tile component before moving to the next step.

Step 5 is performed for all values of *n* such that $i_0 \le 2n + 1 < i_1$. Step 6 is performed for all values of *n* such that $i_0 \le 2n < i_1$.

$$\begin{cases} Y(2n+1) \leftarrow X_{ext}(2n+1) + \alpha \cdot (X_{ext}(2n) + X_{ext}(2n+2)) & [STEP1] \\ Y(2n) \leftarrow X_{ext}(2n) + \beta \cdot (Y(2n-1) + Y(2n+1)) & [STEP2] \\ Y(2n+1) \leftarrow Y(2n+1) + \gamma \cdot (Y(2n) + Y(2n+2)) & [STEP3] \\ Y(2n) \leftarrow Y(2n) + \delta \cdot (Y(2n-1) + Y(2n+1)) & [STEP4] \\ Y(2n+1) \leftarrow K \cdot Y(2n+1) & [STEP5] \\ Y(2n) \leftarrow (1/K) \cdot Y(2n) & [STEP6] \end{cases} \quad \text{F.5}$$

where the values of the parameters $(\alpha, \beta, \gamma, \delta)$ are:

$$\begin{cases} \alpha = -1{,}586134342 \\ \beta = -0{,}05298011854 \\ \gamma = 0{,}8829110762 \\ \delta = 0{,}4435068522 \end{cases} \quad \text{F.6}$$

and the scaling factor *K* is equal to: *K*=1,149604

F.2.8.2 Reversible 1D filtering

The reversible transformation described in this section is the reversible lifting-based implementation of filtering by the 5/3 filter [11].

The reversible transformation is also described using lifting-based filtering. Reversible lifting-based filtering consists of a sequence of simple filtering operations for which alternately, odd sample values of the signal are updated with a weighted sum of even sample values which is rounded to an integer value, and even sample values are updated with a weighted sum of odd sample values which is rounded to an integer value.

The odd samples of output signal *Y* are computed first for all values of *n* such that $-1 \le 2n + 1 < i_1 + 1$:

$$(2n+1) \;=\; X_{ext}(2n+1) - \left\lceil \frac{X_{ext}(2n) + X_{ext}(2n+2) - 1}{2} \right\rceil. \qquad \text{F.7}$$

Then the even samples of output signal *Y* are computed from the even values of extended signal X_{ext} and the odd samples of signal *Y* for all values of *n* such that $0 \le 2n < i_1$

$$(2n) \;=\; X_{ext}(2n) + \left\lfloor \frac{Y(2n-1) + Y(2n+1) + 2}{4} \right\rfloor \qquad \text{F.8}$$

F.2.9 Grouping transform coefficients into sub-bands

As illustrated in Figure F-16, the 2D_ARRANGE procedure rearranges the samples of *J(u,v)* into a number of sub-bands $a_b(u,v)$.

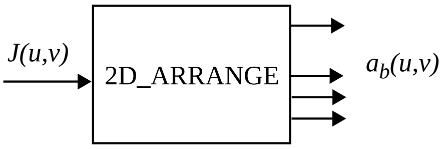


Figure F-16 — Parameters of 2D_ARRANGE procedure

The total number of sub-bands is $3.N_L$+1. The sub-bands are labelled in the following way: an index *lev* corresponding to the level of the sub-band decomposition, followed by two letters which are either LL, LH, HL or HH. Coefficients from the sub-band b=*lev*LH, are the transform coefficients obtained from low-pass filtering horizontally and high-pass filtering vertically at decomposition level *lev.* Coefficients from the sub-band b=*lev*HL, are the transform coefficients obtained from high-pass filtering horizontally and low-pass filtering vertically at decomposition level *lev.* Coefficients from the sub-band b=*lev*HH, are the transform coefficients obtained from high-pass filtering horizontally and high-pass filtering vertically at decomposition level *lev.* Coefficients from the sub-band b=N_LLL, are the transform coefficients obtained from low-pass filtering horizontally and low-pass filtering vertically at the last decomposition level N_L.

For some functions a sub-band order is used. This is defined as the following:

$\mathrm{N_L}$LL, ($\mathrm{N_L}$-1)HL, ($\mathrm{N_L}$-1)LH, ($\mathrm{N_L}$-1)HH, ... , 1HL, 1LH, 1HH

The way these sub-bands are formed from the output *J(u,v)* of the FDWT procedure is described by the 2D_ARRANGE procedure illustrated in Figure F-17. The formation of the sub-bands is simply a reordering of the samples of *J(u,v).*

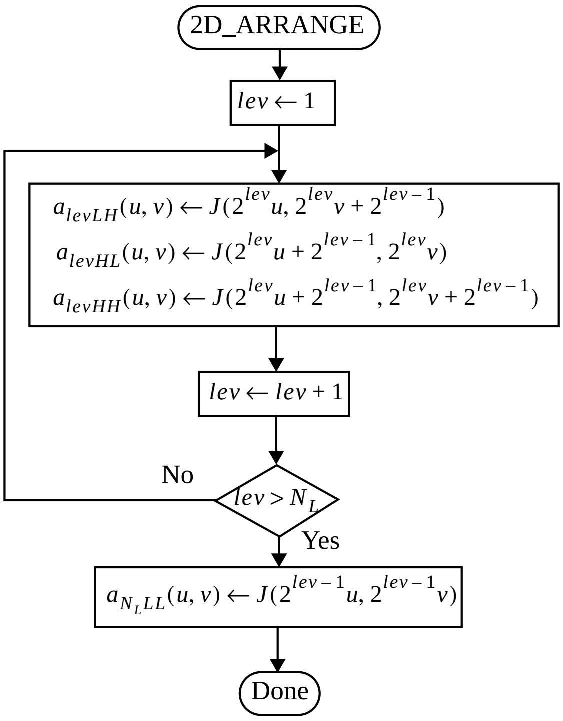

2D_ARRANGE
$lev \leftarrow 1$
$a_{levLH}(u,v) \leftarrow J(2^{lev}u, 2^{lev}v + 2^{lev-1})$
$a_{levHL}(u,v) \leftarrow J(2^{lev}u + 2^{lev-1}, 2^{lev}v)$
$a_{levHH}(u,v) \leftarrow J(2^{lev}u + 2^{lev-1}, 2^{lev}v + 2^{lev-1})$
$lev \leftarrow lev + 1$
No
$lev > N_L$
Yes
$a_{N_LLL}(u,v) \leftarrow J(2^{lev-1}u, 2^{lev-1}v)$
Done

Figure F-17 — The 2D_ARRANGE procedure

As illustrated in Figure F-18, all the sub-bands in the case where N_L=2 can be represented in the following way:

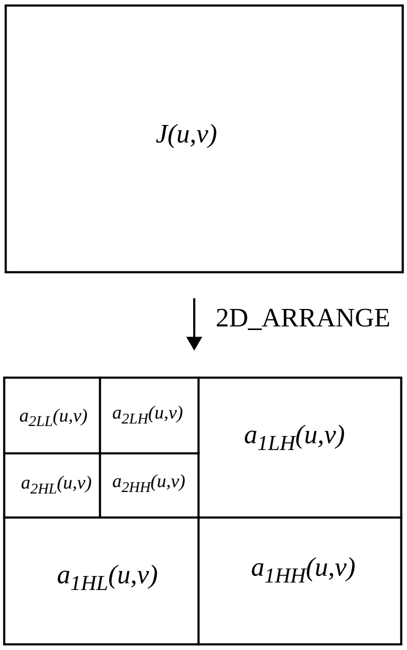


Figure F-18 — Arrangement of transform coefficients into sub-bands (N_L=2)

F.3 The inverse transformation (normative)

F.3.1 Rearranging sub-band samples into transform samples

As illustrated in Figure F-19, the 2D_REARRANGE procedure rearranges the sub-bands samples *$a_b(u,v)$* into the transform samples *J(u,v)*.

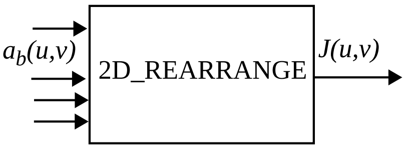


Figure F-19 — Parameters of the 2D_REARRANGE procedure

As illustrated in Figure F-20, the way this rearrangement is performed is described by the 2D_REARRANGE procedure described below. The formation of the transform samples of *J(u,v)* is simply a re-arrangement of the samples from the various sub-bands.

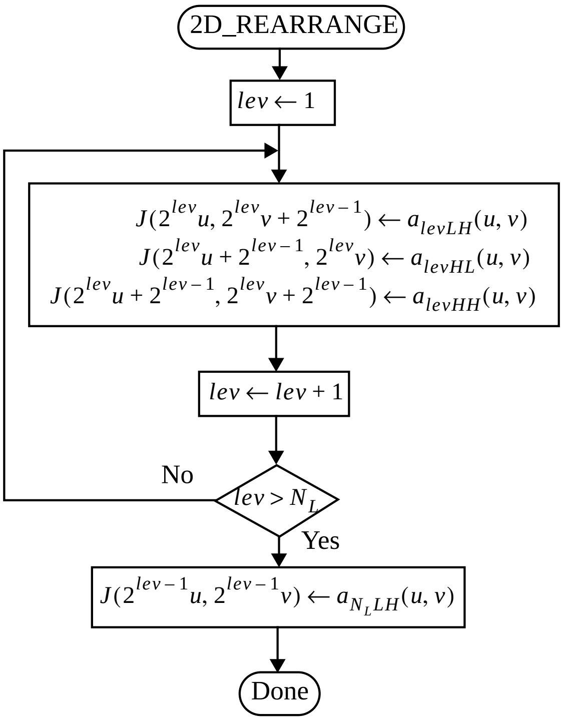

2D_REARRANGE
$lev \leftarrow 1$
$J(2^{lev}u, 2^{lev}v + 2^{lev-1}) \leftarrow a_{levLH}(u, v)$
$J(2^{lev}u + 2^{lev-1}, 2^{lev}v) \leftarrow a_{levHL}(u, v)$
$J(2^{lev}u + 2^{lev-1}, 2^{lev}v + 2^{lev-1}) \leftarrow a_{levHH}(u, v)$
$lev \leftarrow lev + 1$
No
$lev > N_L$
Yes
$J(2^{lev-1}u, 2^{lev-1}v) \leftarrow a_{N_LLH}(u, v)$
Done

Figure F-20 — The 2D_REARRANGE procedure

F.3.2 The IDWT procedure

As illustrated in Figure F-21, the IDWT takes as input the transformed image tile component samples *J(u,v)* and produces as output the level shifted image tile component samples *I'(x,y)*, where N_L is a parameter of the IDWT representing a number of iterations.

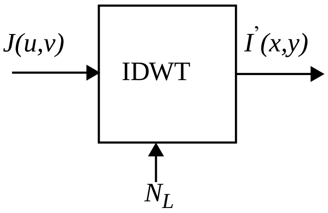


Figure F-21 — Parameters of the IDWT procedure

As illustrated in Figure F-22, the IDWT procedure starts with the initialization of the variable *lev* to N_L, and the initialization the samples *I'(x,y)* to those of *J(u,v)*. The 2D_SR procedure is performed at level *lev*, where the level *lev* decreases at each iteration, and until N_L iterations are performed.

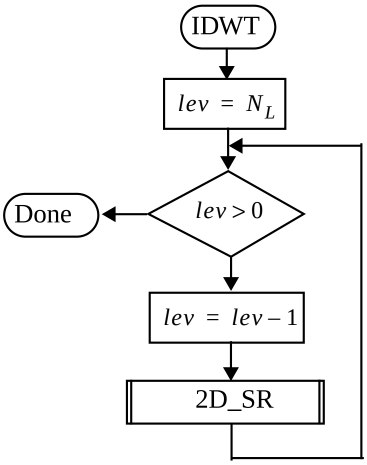


Figure F-22 — The IDWT procedure

F.3.3 The 2D_SR procedure

As illustrated in Figure F-23, the 2D_SR (2D sub-band recomposition) procedure modifies a portion of the samples *J(u,v)*, according to the value of the variable *lev*.

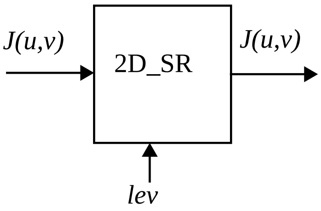


Figure F-23 — Parameters of the 2D_SR procedure

As illustrated in Figure F-24, it begins by applying the HOR_SR procedure to all rows of *J(u,v)* for which *v* is a multiple of $2^{lev\text{-}1}$. It then applies the VER_SR procedure to all columns of *J(u,v)* for which *u* is a multiple of $2^{lev\text{-}1}$.

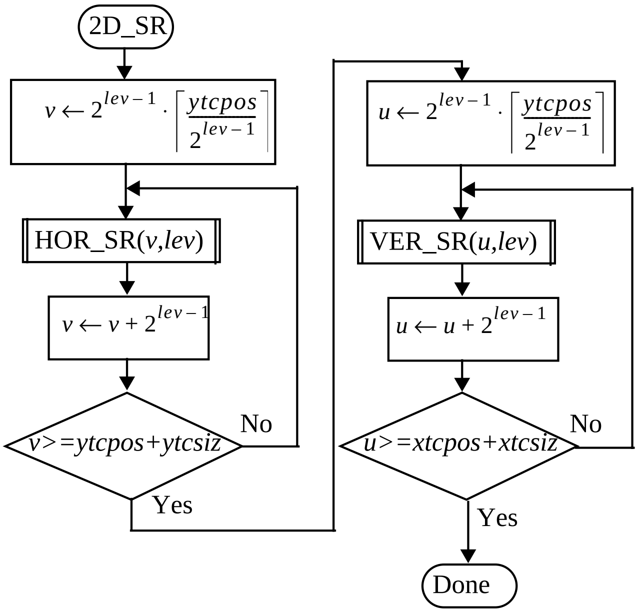

2D_SR
$v \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
HOR_SR(*v*,*lev*)
$v \leftarrow v + 2^{lev-1}$
v>=*ytcpos*+*ytcsiz*
No
Yes
$u \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
VER_SR(*u*,*lev*)
$u \leftarrow u + 2^{lev-1}$
u>=*xtcpos*+*xtcsiz*
No
Yes
Done

Figure F-24 — The 2D_SR procedure

F.3.4 The HOR_SR procedure

As illustrated in Figure F-25, the HOR_SR procedure modifies the sample values of row *v* of *J(u,v)*, for which *u* is a multiple of $2^{lev\text{-}1}$.

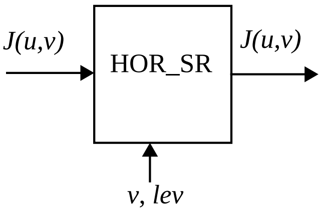


Figure F-25 — Parameters of the HOR_SD procedure

As illustrated in Figure F-26, the HOR_SR procedure stores all the sample values of row *v* of *J(u,v)*, for which *u* is a multiple of 2^{lev-1} into an array *X*, applies the 1D_SR procedure to array *X* and produces array *Y*. Then the values of array *Y* are replaced into the sample values of row *v* of *J(u,v)*, for which *u* is a multiple of 2^{lev-1}.

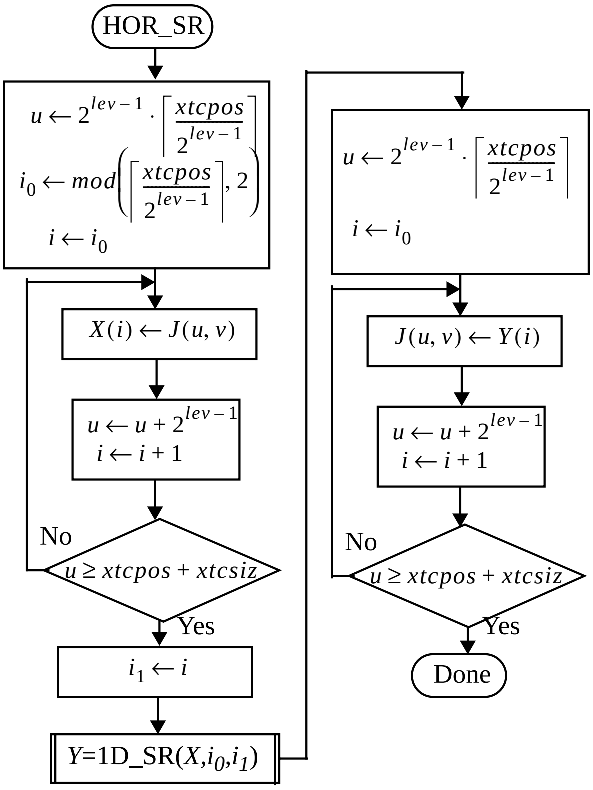

HOR_SR
$u \leftarrow 2^{lev-1} \cdot \left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil$
$i_0 \leftarrow mod\left(\left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil, 2\right)$
$i \leftarrow i_0$
$X(i) \leftarrow J(u, v)$
$u \leftarrow u + 2^{lev-1}$
$i \leftarrow i + 1$
No
$u \geq xtcpos + xtcsiz$
Yes
$i_1 \leftarrow i$
Y=1D_SR(X,i_0,i_1)
$u \leftarrow 2^{lev-1} \cdot \left\lceil \frac{xtcpos}{2^{lev-1}} \right\rceil$
$i \leftarrow i_0$
$J(u, v) \leftarrow Y(i)$
$u \leftarrow u + 2^{lev-1}$
$i \leftarrow i + 1$
No
$u \geq xtcpos + xtcsiz$
Yes
Done

Figure F-26 — The HOR_SR procedure

F.3.5 The VER_SR procedure

As illustrated in Figure F-27, the VER_SR procedure modifies the sample values of column *u* of $J_{t,c}(u,v)$, for which *v* is a multiple of 2^{lev-1}.

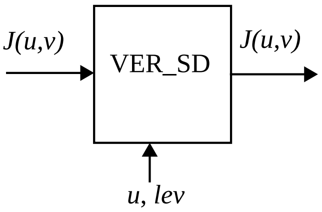


Figure F-27 — Parameters of the VER_SD procedure

As illustrated in Figure F-28, the VER_SR procedure stores all the sample values of column *u* of *J(u,v)*, for which *v* is a multiple of 2^{lev-1} into an array *X*, applies the 1D_SR procedure to array *X* and produces array *Y*. Then the values of array *Y* are replaced into the sample values of column *u* of *J(u,v)*, for which *v* is a multiple of 2^{lev-1}.

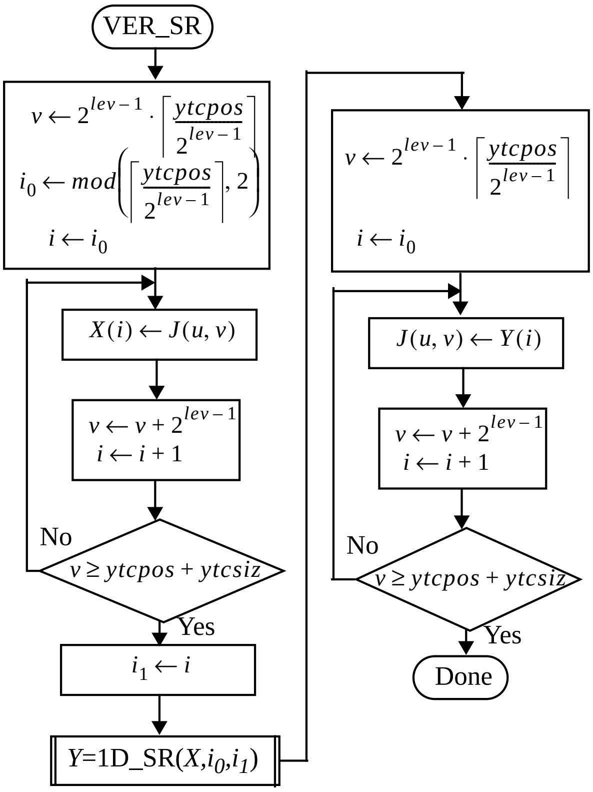

VER_SR
$v \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
$i_0 \leftarrow mod\left(\left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil, 2\right)$
$i \leftarrow i_0$
$X(i) \leftarrow J(u, v)$
$v \leftarrow v + 2^{lev-1}$
$i \leftarrow i + 1$
No
$v \geq ytcpos + ytcsiz$
Yes
$i_1 \leftarrow i$
Y=1D_SR(X,i_0,i_1)
$v \leftarrow 2^{lev-1} \cdot \left\lceil \frac{ytcpos}{2^{lev-1}} \right\rceil$
$i \leftarrow i_0$
$J(u, v) \leftarrow Y(i)$
$v \leftarrow v + 2^{lev-1}$
$i \leftarrow i + 1$
No
$v \geq ytcpos + ytcsiz$
Yes
Done

Figure F-28 — The VER_SR procedure

F.3.6 The 1D_SR procedure

As illustrated in Figure F-29, the 1D_SR procedure takes as input an array *Y*, the index i_0 (which has value of 0 or 1) to the first sample of array *Y*, and an index i_1 to the sample following the last sample value of array *Y*. Both i_0 and *i* are calculated by the VER_SR or HOR_SR procedure which is calling the 1D_SR procedure.

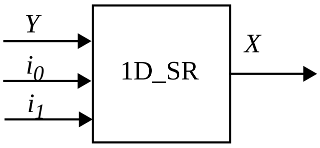


Figure F-29 — Parameters of the 1D_SR procedure

As illustrated in Figure F-30, the 1D_SR procedure first uses the 1D_EXT procedure to extend the signal *Y* beyond its left and right boundaries resulting in the extended signal Y_{ext}, and then uses the 1D_IFILT procedure to inverse filter the extended signal Y_{ext} and produce the desired filtered signal *X*.

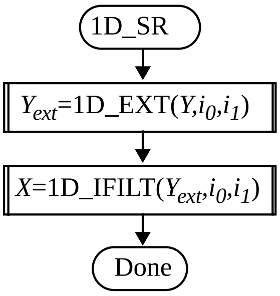

1D_SR
Y_{ext}=1D_EXT(Y,i_0,i_1)
X=1D_IFILT(Y_{ext},i_0,i_1)
Done

Figure F-30 — The 1D_SR procedure

F.3.7 The 1D_EXT procedure

The 1D_EXT procedure is identical to the one described for the forward transformation.

F.3.8 The 1D_IFILT procedure

One irreversible inverse filtering procedure 1D_IFILT_I and one reversible filtering procedure 1D_IFILT_R are described.

As illustrated in Figure F-31, both procedures take as input an extended 1D signal Y_{ext}, the index of the first sample i_0, and the index of the sample i_1 immediately following the last sample (i_1-1). They both produce as output signal *X*.

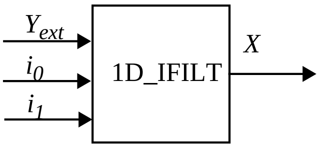


Figure F-31 — Parameters of the 1D_IFILT procedure

Both the irreversible and reversible inverse transformations are described using lifting-based inverse filtering [15], which is a very efficient implementation of the inverse DWT. Lifting-based filtering consists of a sequence of very simple filtering operations for which alternately, odd sample values of the signal are updated with a weighted sum of even sample values, and even sample values are updated with a weighted sum of odd sample values.

F.3.8.1 Irreversible 1D inverse filtering

The irreversible inverse transformation described in this section is the lifting-based DWT implementation of filtering by the Daubechies 9/7 filter [4].

Equation F.9 describes the 2 “scaling” steps (1 and 2) and the 4 “lifting” steps (3 through 6) and of the 1D filtering performed on the extended signal $Y_{ext}(n)$ to produce the i_1 samples of signal *X*.

Step 1 is performed for all values of *n* such that $i_0 - 3 \le 2n < i_1 + 3$ and step 2 is performed for all values of *n* such that $i_0 - 2 \le 2n < i_1 + 2$.

Step 3 is performed for all values of *n* such that $i_0 - 3 \le 2n < i_1 + 3$. Step 4 is performed for all values of *n* such that $i_0 - 2 \le 2n + 1 < i_1 + 2$. Step 5 is performed for all values of *n* such that $i_0 - 1 \le 2n < i_1 + 1$. Finally, step 6 is performed for all values of *n* such that $i_0 \le 2n + 1 < i_1$.

$$\begin{cases} X(2n) \leftarrow K \cdot Y_{ext}(2n) & [STEP1] \\ X(2n+1) \leftarrow (1/K) \cdot Y_{ext}(2n+1) & [STEP2] \\ X(2n) \leftarrow X(2n) - \delta \cdot (X(2n-1) + X(2n+1)) & [STEP3] \\ X(2n+1) \leftarrow X(2n+1) - \gamma \cdot (X(2n) + X(2n+2)) & [STEP4] \\ X(2n) \leftarrow X(2n) - \beta \cdot (X(2n-1) + X(2n+1)) & [STEP5] \\ X(2n+1) \leftarrow X(2n+1) - \alpha \cdot (X(2n) + X(2n+2)) & [STEP6] \end{cases} \qquad \text{F.9}$$

where the values of the parameters $(\alpha, \beta, \gamma, \delta)$ are:

$$\begin{cases} \alpha = -1{,}586134342 \\ \beta = -0{,}05298011854 \\ \gamma = 0{,}8829110762 \\ \delta = 0{,}4435068522 \end{cases} \qquad \text{F.10}$$

and the scaling factor *K* is equal to: *K*=1,149604.

F.3.8.2 Reversible 1D inverse filtering

The reversible inverse transformation is also described using lifting-based filtering. Reversible lifting-based inverse filtering consists of a sequence of simple filtering operations for which alternately, odd sample values of the signal are updated with a weighted sum of even sample values which is rounded to an integer value, and even sample values are updated with a weighted sum of odd sample values which is rounded to an integer value.

The even sample values of output signal *X* are computed first from the even sample values of extended signal Y_{ext} and the odd sample values of signal Y_{ext} for all values of *n* such that $i_0 - 1 \le 2n < i_1 - 1$:

$$(2n) = Y_{ext}(2n) - \left\lfloor \frac{Y_{ext}(2n-1) + Y_{ext}(2n+1) + 2}{4} \right\rfloor \qquad \text{F.11}$$

Then the odd sample values of output signal *X* are computed from the odd sample values of extended signal Y_{ext} and the even sample values of signal Y_{ext} for all values of *n* such that $i_0 \le 2n + 1 < i_1$:

$$(2n+1) = Y_{ext}(2n+1) + \left\lceil \frac{Y_{ext}(2n) + Y_{ext}(2n+2) - 1}{2} \right\rceil . \qquad \text{F.12}$$

F.3.9 DC level shifting of image tile components

After computation of the IDWT, if the MSB of Ssiz is zero, then all samples $I'(x,y)$ of image tile component *c* are inverse level shifted to their original representation by adding the same quantity from each sample, where $Ssiz^c$ is the component's precision parameter specified in Annex A:

$$I(x, y) = I'(x, y) + (2^{Ssiz^c - 1}) \quad . \qquad \text{F.13}$$

NOTE — Due to quantization, the reconstructed sample values I(x,y) may exceed the dynamic range of the original image tile component sample values.

F.4 Row-based wavelet transform (informative)

Described here is a row-based wavelet transform well suited for compression devices which received and transferred image data in a serial manner. Traditional wavelet transform implementations require the whole image to be buffered and filtering to be performed in vertical and horizontal directions. While filtering in the horizontal direction is very simple, filtering in the vertical direction is more involved. Filtering along a row requires one row to be read; filtering along a column requires the whole image to be read. This explains the huge bandwidth requirements of the traditional wavelet transform implementation. The row-based wavelet transform overcomes the previous limitation while providing the exact same transformed coefficients as traditional wavelet transform implementation. However, it has to be underlined that the row-based wavelet transform alone does not provide a complete row-based encoding paradigm. A complete row-based coder has to take also into account all the following coding stage up to the entropy coding stage.

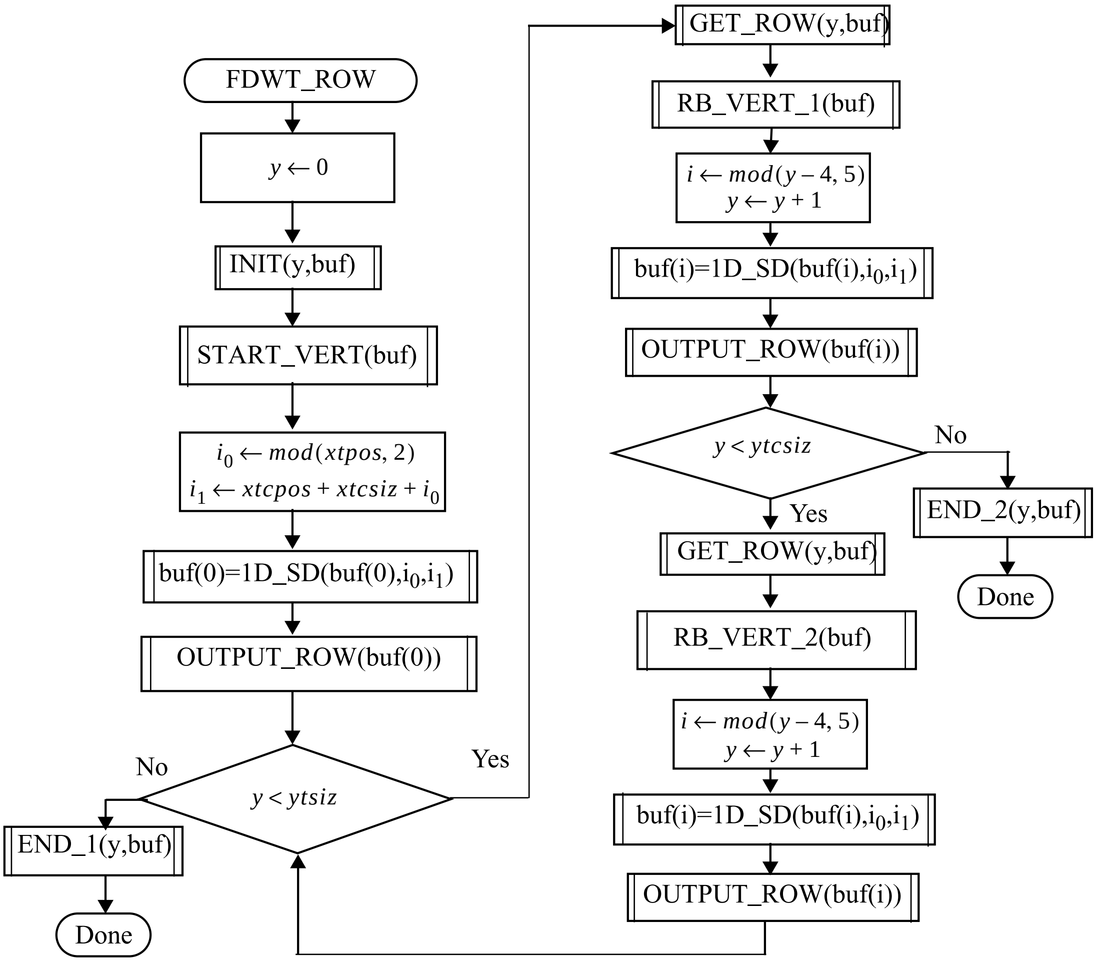


Figure F-32 — The FDWT_ROW procedure

F.4.1 The FDWT_ROW procedure

The FDWT_ROW procedure uses one buffer buf(i,j) of five lines, $0 \le i \le 4$, for performing a one level wavelet decomposition on one row of length ytsiz in the vertical direction for the 9/7 wavelet filter. Each line of the buffer buf(i,j) is of size xtsiz+1. The general description of the FDWT_ROW applied to one image tile component is illustrated in Figure F-32 for the first level of decomposition. The FDWT_ROW takes as input level shifted image tile component line of samples and produces as output one line of transform coefficients. It is assumed throughout this section that the image tile component has at least five rows.

F.4.1.1 The GET_ROW procedure

In this description, the level shifted image tile component is supposed to be stored in an external memory $I'(x, y)$. As illustrated in Figure F-33, the GET_ROW procedure reads one line of samples of the level shifted image tile component and transfer this line of samples in the buffer buf.

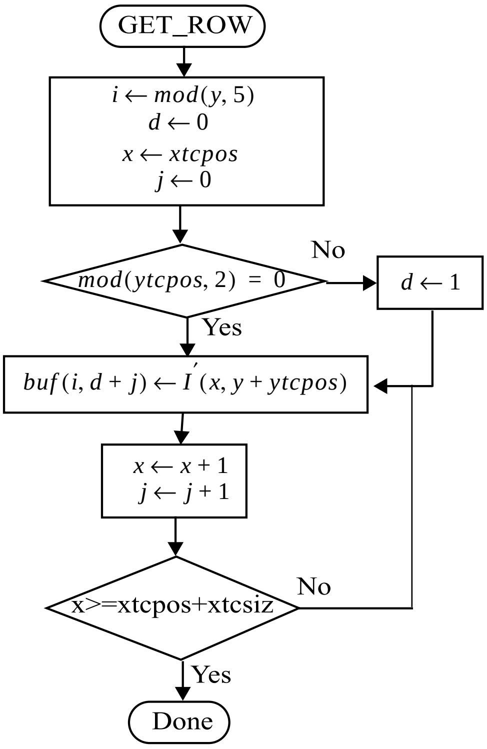


Figure F-33 — The GET_ROW procedure

F.4.2 The INIT procedure

As illustrated in Figure F-34, the INIT procedure reads five lines of samples of the level shifted image tile component and transfer these lines of samples in the buffer, buf.

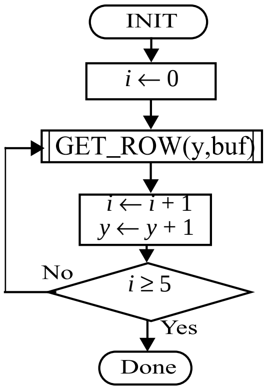


Figure F-34 — The INIT procedure

F.4.3 The START_VERT procedure

As illustrated in Figure F-35, the START_VERT procedure modifies the coefficients in the buffer buf(i,j). In this Figure as well as in all the following Figure of this section, the expression $buf(i) \leftarrow buf(i) + \alpha \cdot buf(i_2)$ is equivalent to $buf(i, j) \leftarrow buf(i, j) + \alpha \cdot buf(i_2, j)$ for $0 \le j < xtcsiz + 1$

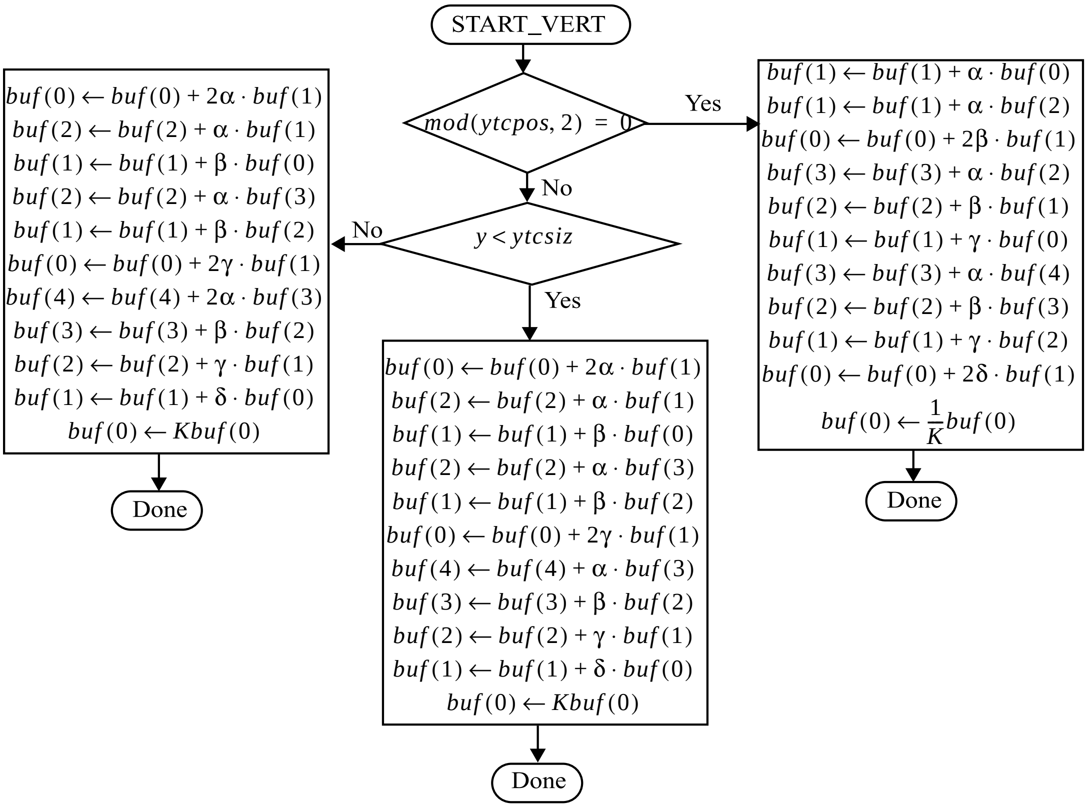

START_VERT
$mod(ytcpos, 2) = 0$
Yes
No
$buf(1) \leftarrow buf(1) + \alpha \cdot buf(0)$
$buf(1) \leftarrow buf(1) + \alpha \cdot buf(2)$
$buf(0) \leftarrow buf(0) + 2\beta \cdot buf(1)$
$buf(3) \leftarrow buf(3) + \alpha \cdot buf(2)$
$buf(2) \leftarrow buf(2) + \beta \cdot buf(1)$
$buf(1) \leftarrow buf(1) + \gamma \cdot buf(0)$
$buf(3) \leftarrow buf(3) + \alpha \cdot buf(4)$
$buf(2) \leftarrow buf(2) + \beta \cdot buf(3)$
$buf(1) \leftarrow buf(1) + \gamma \cdot buf(2)$
$buf(0) \leftarrow buf(0) + 2\delta \cdot buf(1)$
$buf(0) \leftarrow \frac{1}{K} buf(0)$
Done
$y < ytcsiz$
No
$buf(0) \leftarrow buf(0) + 2\alpha \cdot buf(1)$
$buf(2) \leftarrow buf(2) + \alpha \cdot buf(1)$
$buf(1) \leftarrow buf(1) + \beta \cdot buf(0)$
$buf(2) \leftarrow buf(2) + \alpha \cdot buf(3)$
$buf(1) \leftarrow buf(1) + \beta \cdot buf(2)$
$buf(0) \leftarrow buf(0) + 2\gamma \cdot buf(1)$
$buf(4) \leftarrow buf(4) + 2\alpha \cdot buf(3)$
$buf(3) \leftarrow buf(3) + \beta \cdot buf(2)$
$buf(2) \leftarrow buf(2) + \gamma \cdot buf(1)$
$buf(1) \leftarrow buf(1) + \delta \cdot buf(0)$
$buf(0) \leftarrow Kbuf(0)$
Done
Yes
$buf(0) \leftarrow buf(0) + 2\alpha \cdot buf(1)$
$buf(2) \leftarrow buf(2) + \alpha \cdot buf(1)$
$buf(1) \leftarrow buf(1) + \beta \cdot buf(0)$
$buf(2) \leftarrow buf(2) + \alpha \cdot buf(3)$
$buf(1) \leftarrow buf(1) + \beta \cdot buf(2)$
$buf(0) \leftarrow buf(0) + 2\gamma \cdot buf(1)$
$buf(4) \leftarrow buf(4) + \alpha \cdot buf(3)$
$buf(3) \leftarrow buf(3) + \beta \cdot buf(2)$
$buf(2) \leftarrow buf(2) + \gamma \cdot buf(1)$
$buf(1) \leftarrow buf(1) + \delta \cdot buf(0)$
$buf(0) \leftarrow Kbuf(0)$
Done

Figure F-35 — The START_VERT procedure

F.4.3.1 The RB_VERT_1 procedure

As illustrated in Figure F-36, the RB_VERT_1 procedure modifies the coefficient in buf(i,j).

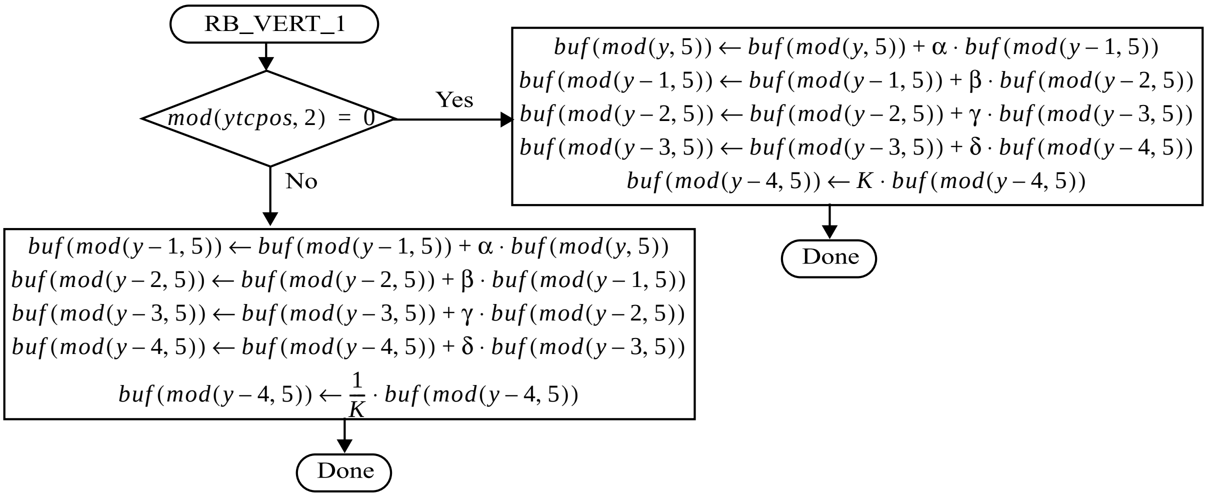

RB_VERT_1
$mod(ytcpos, 2) = 0$
Yes
No
$buf(mod(y, 5)) \leftarrow buf(mod(y, 5)) + \alpha \cdot buf(mod(y-1, 5))$
$buf(mod(y-1, 5)) \leftarrow buf(mod(y-1, 5)) + \beta \cdot buf(mod(y-2, 5))$
$buf(mod(y-2, 5)) \leftarrow buf(mod(y-2, 5)) + \gamma \cdot buf(mod(y-3, 5))$
$buf(mod(y-3, 5)) \leftarrow buf(mod(y-3, 5)) + \delta \cdot buf(mod(y-4, 5))$
$buf(mod(y-4, 5)) \leftarrow K \cdot buf(mod(y-4, 5))$
Done
$buf(mod(y-1, 5)) \leftarrow buf(mod(y-1, 5)) + \alpha \cdot buf(mod(y, 5))$
$buf(mod(y-2, 5)) \leftarrow buf(mod(y-2, 5)) + \beta \cdot buf(mod(y-1, 5))$
$buf(mod(y-3, 5)) \leftarrow buf(mod(y-3, 5)) + \gamma \cdot buf(mod(y-2, 5))$
$buf(mod(y-4, 5)) \leftarrow buf(mod(y-4, 5)) + \delta \cdot buf(mod(y-3, 5))$
$buf(mod(y-4, 5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-4, 5))$
Done

Figure F-36 — The RB_VERT_1 procedure

F.4.3.2 The RB_VERT_2 procedure

As illustrated in Figure F-37, the RB_VERT_2 procedure modifies the coefficient in buf(i,j).

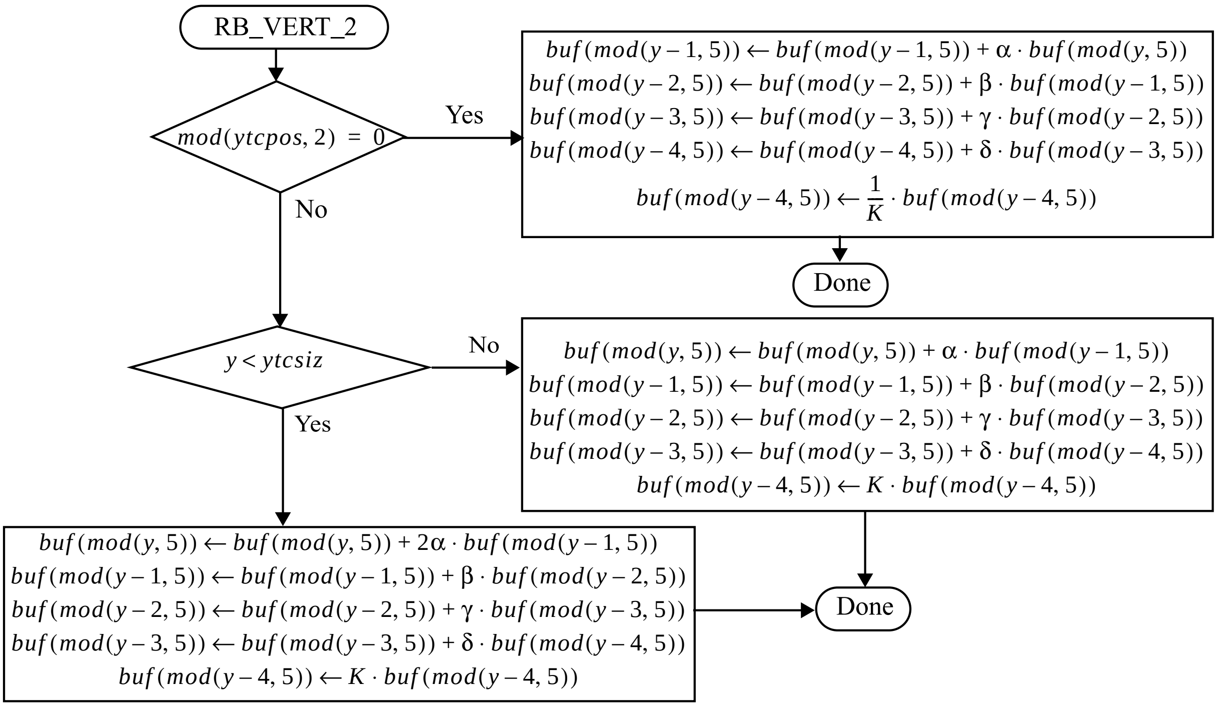

RB_VERT_2
$mod(ytcpos, 2) = 0$
Yes
No
$buf(mod(y-1, 5)) \leftarrow buf(mod(y-1, 5)) + \alpha \cdot buf(mod(y, 5))$
$buf(mod(y-2, 5)) \leftarrow buf(mod(y-2, 5)) + \beta \cdot buf(mod(y-1, 5))$
$buf(mod(y-3, 5)) \leftarrow buf(mod(y-3, 5)) + \gamma \cdot buf(mod(y-2, 5))$
$buf(mod(y-4, 5)) \leftarrow buf(mod(y-4, 5)) + \delta \cdot buf(mod(y-3, 5))$
$buf(mod(y-4, 5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-4, 5))$
Done
$y < ytcsiz$
No
Yes
$buf(mod(y, 5)) \leftarrow buf(mod(y, 5)) + \alpha \cdot buf(mod(y-1, 5))$
$buf(mod(y-1, 5)) \leftarrow buf(mod(y-1, 5)) + \beta \cdot buf(mod(y-2, 5))$
$buf(mod(y-2, 5)) \leftarrow buf(mod(y-2, 5)) + \gamma \cdot buf(mod(y-3, 5))$
$buf(mod(y-3, 5)) \leftarrow buf(mod(y-3, 5)) + \delta \cdot buf(mod(y-4, 5))$
$buf(mod(y-4, 5)) \leftarrow K \cdot buf(mod(y-4, 5))$
$buf(mod(y, 5)) \leftarrow buf(mod(y, 5)) + 2\alpha \cdot buf(mod(y-1, 5))$
$buf(mod(y-1, 5)) \leftarrow buf(mod(y-1, 5)) + \beta \cdot buf(mod(y-2, 5))$
$buf(mod(y-2, 5)) \leftarrow buf(mod(y-2, 5)) + \gamma \cdot buf(mod(y-3, 5))$
$buf(mod(y-3, 5)) \leftarrow buf(mod(y-3, 5)) + \delta \cdot buf(mod(y-4, 5))$
$buf(mod(y-4, 5)) \leftarrow K \cdot buf(mod(y-4, 5))$
Done

Figure F-37 — The RB_VERT_2 procedure

F.4.3.3 The END_1 procedure

The END_1 procedure is detailed in Figure F-38.

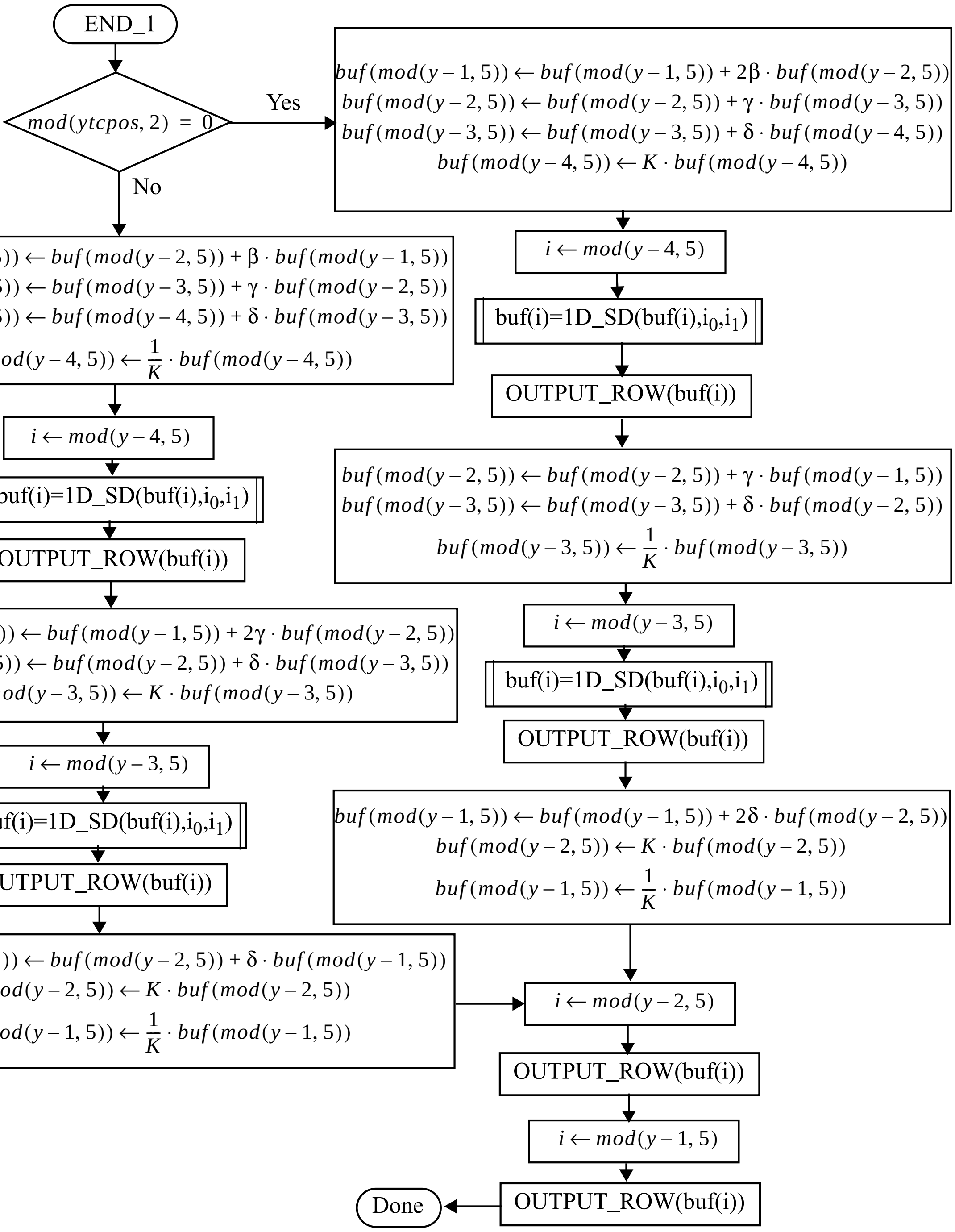

END_1
$mod(ytcpos, 2) = 0$
Yes
No
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + 2\beta \cdot buf(mod(y-2,5))$
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \gamma \cdot buf(mod(y-3,5))$
$buf(mod(y-3,5)) \leftarrow buf(mod(y-3,5)) + \delta \cdot buf(mod(y-4,5))$
$buf(mod(y-4,5)) \leftarrow K \cdot buf(mod(y-4,5))$
$i \leftarrow mod(y-4,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \gamma \cdot buf(mod(y-1,5))$
$buf(mod(y-3,5)) \leftarrow buf(mod(y-3,5)) + \delta \cdot buf(mod(y-2,5))$
$buf(mod(y-3,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-3,5))$
$i \leftarrow mod(y-3,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + 2\delta \cdot buf(mod(y-2,5))$
$buf(mod(y-2,5)) \leftarrow K \cdot buf(mod(y-2,5))$
$buf(mod(y-1,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-1,5))$
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \beta \cdot buf(mod(y-1,5))$
$buf(mod(y-3,5)) \leftarrow buf(mod(y-3,5)) + \gamma \cdot buf(mod(y-2,5))$
$buf(mod(y-4,5)) \leftarrow buf(mod(y-4,5)) + \delta \cdot buf(mod(y-3,5))$
$buf(mod(y-4,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-4,5))$
$i \leftarrow mod(y-4,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + 2\gamma \cdot buf(mod(y-2,5))$
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \delta \cdot buf(mod(y-3,5))$
$buf(mod(y-3,5)) \leftarrow K \cdot buf(mod(y-3,5))$
$i \leftarrow mod(y-3,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \delta \cdot buf(mod(y-1,5))$
$buf(mod(y-2,5)) \leftarrow K \cdot buf(mod(y-2,5))$
$buf(mod(y-1,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-1,5))$
$i \leftarrow mod(y-2,5)$
OUTPUT_ROW(buf(i))
$i \leftarrow mod(y-1,5)$
OUTPUT_ROW(buf(i))
Done

Figure F-38 — The END_1 procedure

F.4.3.4 The END_2 procedure

The END_2 procedure is detailed in Figure F-39

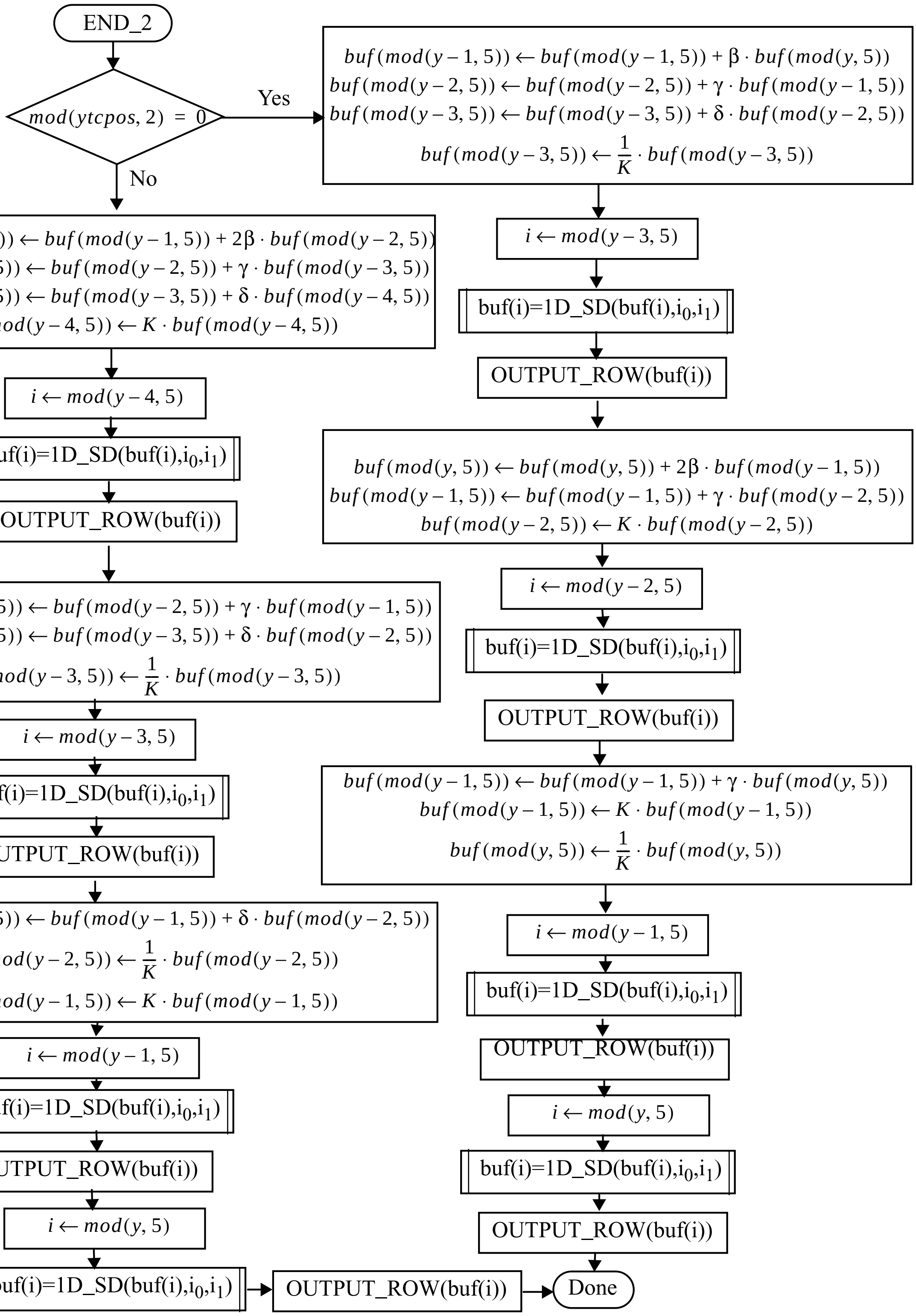

END_2
$mod(ytcpos, 2) = 0$
Yes
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + \beta \cdot buf(mod(y,5))$
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \gamma \cdot buf(mod(y-1,5))$
$buf(mod(y-3,5)) \leftarrow buf(mod(y-3,5)) + \delta \cdot buf(mod(y-2,5))$
$buf(mod(y-3,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-3,5))$
$i \leftarrow mod(y-3,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y,5)) \leftarrow buf(mod(y,5)) + 2\beta \cdot buf(mod(y-1,5))$
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + \gamma \cdot buf(mod(y-2,5))$
$buf(mod(y-2,5)) \leftarrow K \cdot buf(mod(y-2,5))$
$i \leftarrow mod(y-2,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + \gamma \cdot buf(mod(y,5))$
$buf(mod(y-1,5)) \leftarrow K \cdot buf(mod(y-1,5))$
$buf(mod(y,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y,5))$
$i \leftarrow mod(y-1,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$i \leftarrow mod(y,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
No
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + 2\beta \cdot buf(mod(y-2,5))$
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \gamma \cdot buf(mod(y-3,5))$
$buf(mod(y-3,5)) \leftarrow buf(mod(y-3,5)) + \delta \cdot buf(mod(y-4,5))$
$buf(mod(y-4,5)) \leftarrow K \cdot buf(mod(y-4,5))$
$i \leftarrow mod(y-4,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-2,5)) \leftarrow buf(mod(y-2,5)) + \gamma \cdot buf(mod(y-1,5))$
$buf(mod(y-3,5)) \leftarrow buf(mod(y-3,5)) + \delta \cdot buf(mod(y-2,5))$
$buf(mod(y-3,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-3,5))$
$i \leftarrow mod(y-3,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$buf(mod(y-1,5)) \leftarrow buf(mod(y-1,5)) + \delta \cdot buf(mod(y-2,5))$
$buf(mod(y-2,5)) \leftarrow \frac{1}{K} \cdot buf(mod(y-2,5))$
$buf(mod(y-1,5)) \leftarrow K \cdot buf(mod(y-1,5))$
$i \leftarrow mod(y-1,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
$i \leftarrow mod(y,5)$
buf(i)=1D_SD(buf(i),i_0,i_1)
OUTPUT_ROW(buf(i))
Done

Figure F-39 — The END_2 procedure

F.4.4 OUTPUT_ROW procedure

This procedure returns a line buf(i) of transformed coefficients, which correspond either to the 1LL and 1HL sub-band or to the 1LH and 1HH sub-band. This line of transform coefficient can be either store in an external memory or processed immediately.

Annex G

Image component transformations

(This annex forms an integral part of this Recommendation | International Standard)

This Annex specifies a reversible and an irreversible component transformation.

G.1 Reversible component transformation (RCT)

The use of the reversible component transformation is signalled in the CSsiz parameter in Table A-10 of Annex A. The RCT is a decorrelating transformation which is applied to the the three first components of an image (indexed as 0, 1 and 2).

G.1.1 The Forward RCT (informative)

Prior to applying the Forward RCT, the image component samples are DC level shifted, if needed (see Annex F), before being transformed as described in Annex F.

The Forward RCT is applied to all image component samples *$I_0(x,y)$, $I_1(x,y)$, $I_2(x,y)$,* corresponding to the first, second, and third component, and produces transform samples *$Y_0(x,y)$, $Y_1(x,y)$, $Y_2(x,y)$*:

$${}_0(x, y) = \left\lfloor \frac{I_0(x, y) + 2I_1(x, y) + I_2(x, y)}{4} \right\rfloor \qquad \text{G.1}$$

$${}_1(x, y) = I_0(x, y) - I_1(x, y) \qquad \text{G.2}$$

$${}_2(x, y) = I_2(x, y) - I_1(x, y) \qquad \text{G.3}$$

NOTE — If the first three components are Red, Green and Blue components, then the Forward RCT can be seen as an approximation of a Yuv transformation.

G.1.2 The Inverse RCT (normative)

After being inverse transformed as described in Annex F, the following transformation is specified to perform the Inverse RCT:

$${}_1(x, y) = Y_0(x, y) - \left\lfloor \frac{Y_1(x, y) + Y_2(x, y)}{4} \right\rfloor \qquad \text{G.4}$$

$${}_0(x, y) = Y_1(x, y) + I_1(x, y) \qquad \text{G.5}$$

$${}_2(x, y) = Y_2(x, y) + I_1(x, y) \qquad \text{G.6}$$

After applying the Inverse RCT, the image component samples are inverse DC level shifted, if needed (see Annex F).

G.2 Irreversible component transformation (ICT).

This section specifies an irreversible component transformation. The use of the irreversible component transformation is signalled in the CSsiz parameter in Table A-10 of Annex A. The ICT is a decorrelating transformation which is applied to the the three first components of an image (indexed as 0, 1 and 2).

G.2.1 The Forward ICT

The Forward ICT is applied to all image component samples $I_0(x,y)$, $I_1(x,y)$, $I_2(x,y)$, corresponding to the first, second, and third component, and produces transform samples $Y_0(x,y)$, $Y_1(x,y)$, $Y_2(x,y)$:

$$ {}_0(x, y) = 0,299 \cdot I_0(x, y) + 0,587 \cdot I_1(x, y) + 0,114 \cdot I_2(x, y) \qquad \text{G.7}$$

$$ {}_1(x, y) = -0,16875 \cdot I_0(x, y) - 0,33126 \cdot I_1(x, y) + 0,5 \cdot I_2(x, y) \qquad \text{G.8}$$

$$ {}_2(x, y) = 0,5 \cdot I_0(x, y) - 0,41869 \cdot I_1(x, y) - 0,08131 \cdot I_2(x, y) \qquad \text{G.9}$$

NOTE — If the first three components are Red, Green and Blue components, then the Forward RCT can be seen as an approximation of a YCbCr transformation.

G.2.2 The Inverse ICT

After being inverse transformed as described in Annex F, the following transformation is specified to perform the Inverse ICT:

$$I_0(x, y) = I_0(x, y) + 1,402 \cdot I_2(x, y) \qquad \text{G.10}$$

$$I_1(x, y) = I_0(x, y) - 0,34413 \cdot I_1(x, y) - 0,71414 \cdot I_2(x, y) \qquad \text{G.11}$$

$$I_2(x, y) = I_0(x, y) + 1,772 \cdot I_1(x, y) \qquad \text{G.12}$$

After applying the Inverse ICT, the image component samples are inverse DC level shifted, if needed (see Annex F).

G.3 Chrominance component sub-sampling and the image reference grid (informative)

The relationship between the components and the reference grid is signalled in the SIZ marker (see Annex A.5.1) and described in Annex D.1.

G.3.1 An example of the interpretation of multiple components

The interpretation of multiple components is unspecified within the scope of the codestream. Interpretations, such as multiple colour components, may be supplied by the file format, the application, or other source.

An example of a non-traditional interpretation is the coding of Regions of interest (ROIs) in a complex SAR data set. Each ROI may be thought of as a set of two image chips representing the real (I) and imaginary (Q) parts of the data. The ensemble of I and Q chips may be assembled into a set of “multiple components,” even though the individual chips are disjoint and may have different spatial dimensions. By-passing the colour space transform, the ensemble of chips may then be subjected to lossless or lossy compression. This procedure has two advantages: all the ROIs in a given data set can be compressed in a single pass; and bit allocation can be optimized across the ensemble of ROIs rather than on a chip-by-chip basis.

Annex H

Coding of images with Regions-of-interest

(This annex forms an integral part of this Recommendation | International Standard)

This annex describes the Region of Interest (ROI) technology. An ROI is a part of an image that is encoded with higher fidelity than the rest of the image (the background). The encoding is also done in such a way that the information associated with the ROI precedes the information associated with the background. The method used (and described in this annex) is the Maxshift method.

H.1 Description of the Maxshift method

H.1.1 Encoding (informative)

The encoding of the quantized transform coefficients is done in a similar way as without any ROI coding. At the encoder side a ROI mask is created describing which quantized transform coefficients must be encoded with better quality (even up to losslessly) in order to encode the ROI with better quality (up to losslessly). The ROI mask is a bit map describing these coefficients. See Annex H.2 for details on how the mask is generated.

The quantized transform coefficients outside of the ROI mask (to be called background coefficients) are scaled down so that the bits associated with the ROI are placed in higher bit-planes than the background. This means that when the entropy coder encodes the quantized transform coefficients, the bit planes associated with the ROI are coded before the information associated with the background. The scaling value used must be sufficiently large to make the smallest non-zero ROI coefficient, $q_{ROI}(x,y)$, larger than the largest background coefficient, $q_{BG}(x,y)$, see Annex H.1.2.

The method can be described using the following steps:

1) Generate ROI mask, M(x,y), see Annex H.2.
2) Find the scaling value, s, seeAnnex H.1.2.
3) Scale down all background coefficients given by M(x,y) using the scaling value s.
4) Write the scaling value s into codestream using the RGN marker described in Annex A.6.3

After these four steps the quantized transform coefficients are entropy coded as usual.

H.1.2 Selection of scaling value, s, at encoder side (normative)

The scaling value, s, must be chosen so that Equation H.1 holds

$$s \geq max(M_b) \qquad \text{H.1}$$

where $max(M_b)$ is the largest number of magnitude bit planes, see Equation E.1, for any background coefficient, $q_{BG}(x,y)$ in any code-block in the current component.

This means that the scaling value used will be sufficiently large to make the smallest non-zero ROI coefficient, $q_{ROI}(x,y)$, larger than the largest background coefficient, $q_{BG}(x,y)$. This, in turn, means that after the scaling of the background coefficients, all significant bits associated with the ROI will be in higher bit planes than all the significant bits associated with the background.

H.1.3 Decoding (normative)

At the decoder side the received quantized coefficients are compared to the threshold value 2^s, where s is the ROI scaling value for this component obtained from the RGN marker in the codestream (see Annex A.6.3). All coefficients that are found to be lower than 2^s are known to belong to the background. These coefficients are scaled up by 2^s.

The method can be described using the following steps:

1) Get the scaling value, s, from the RGN marker in the codestream, see Annex A.6.3.

2) Compare each quantized transform coefficient q(x,y) to 2^s. If the coefficient is below 2^s scale up the coefficient by 2^s

H.2 Region of interest mask generation

To achieve an ROI with better quality than the rest of the image while maintaining a fair amount of compression, bits need to be saved by sending less information for the background. To do this an ROI mask is calculated. The mask is a bit plane indicating a set of quantized transform coefficients whose coding is sufficient in order for the receiver to reconstruct the desired region with better quality than the background (up to lossless). This mask denotes all coefficients that are needed in order to reconstruct the ROI.

To illustrate the concept of ROI mask generation, let us restrict ourselves to a single ROI and a single image component, and identify the pixels which belong to the ROI in the image domain by a binary mask, M[m,n], where

$$M(x,y) = \begin{cases} 1 & \text{wavelet coefficient (x,y) is needed} \\ 0 & \text{accuracy on (x,y) can be sacrificed without affecting ROI} \end{cases} \qquad \text{H.2}$$

M is then bit-wise 1 for all ROI coefficients so that if the first bit of M is 1 then M(x, y) belongs to the first ROI.

The mask is derived following the same steps as the forward transform (or actually tracing the inverse transform). The mask thereby gets the same sub-band structure as the wavelet transform. To start out with, the mask is a map of the ROI in the image domain, so that it has a non-zero value inside the ROI and 0 outside. The LL sub-band of the mask is then in each step updated line by line and then column by column. The mask will then indicate which coefficients are needed at this step so that the inverse transform will reproduce the coefficients of the previous mask.

For example, the last step of the inverse transform is a composition of two sub-bands into one. Then to trace this step backwards, the coefficients in the two sub-bands that are needed, are found. The step before that is a composition of four sub-bands into two. To trace this step backwards, the coefficients in the four sub-bands that are needed to give a perfect reconstruction of the coefficients included in the mask for two sub-bands are found.

All steps are then traced backwards to give the mask. If the coefficients corresponding to the mask are transmitted and received, and the inverse transform calculated on them, the desired ROI will be reconstructed with better quality than the rest of the image (up to lossless if the ROI coefficients were coded lossless).

Given below is a description of how the expansion of the mask is acquired from the various filters. Similar method can be used for other filters. Please refer to [25][26][27][28] for more details.

H.2.1 Region of Interest mask generation using the W5X3 filter (informative)

Successful decoding does not depend upon the selection of samples to be scaled. In order to get the optimal set of quantized coefficients to be scaled, the following equations described in this section should be used.

To see what coefficients need to be in the mask, the inverse transform is studied:

$$\begin{aligned} X(2n) &= L(n) - \frac{H(n-1)+H(n)}{4} \\ X(2n+1) &= \frac{L(n)+L(n+1)}{2} + \frac{-H(n-1)+6H(n)-H(n+1)}{8} \end{aligned} \qquad \text{H.3}$$

The coefficients needed to reconstruct X(2n) and X(2n+1) up to lossless can immediately be seen to be L(n), L(n+1), H(n-1), H(n), H(n+1). Hence if X(2n) or X(2n+1) are in the ROI, the listed L:s and H:s are in the mask. Figure H-1 shows that the coefficients needed to reconstruct X(2n) and X(2n+1) lossless are L(n), L(n+1), H(n-1), H(n), H(n+1). Notice that X(2n) and X(2n+1) are even and odd indexed points respectively, related to the partition reference point.

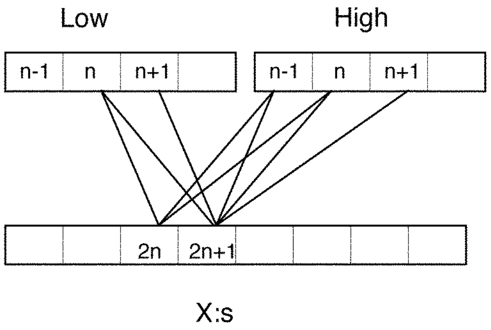


Figure H-1 — The inverse 5x3 transform

H.2.2 Region of Interest mask generation using the W9X7 filter (informative)

Successful decoding does not depend upon the selection of samples to be scaled. In order to get the optimal set of quantized coefficients to be scaled the following equations described in this section should be used.

To see what coefficients need to be in the mask, the inverse transform is studied in similar manner as for the filters above. Figure H-2 shows this. Notice that X(2n) and X(2n+1) are even and odd indexed points respectively, related to the partition reference point

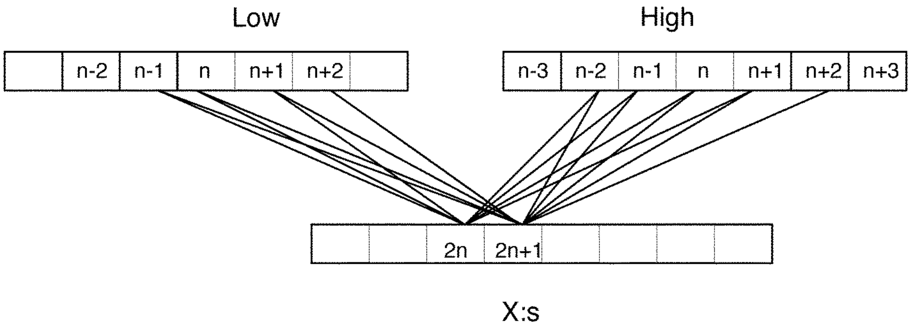


Figure H-2 — The inverse 9x7 transform

The coefficients needed to reconstruct X(2n) and X(2n+1) lossless can immediately be seen to be L(n-1) to L(n+2) and H(n-2) to H(n+2). Hence if X(2n) or X(2n+1) are in the ROI, those L:s and H:s are in the mask

H.3 Remarks on Region of Interest coding

H.3.1 Multi-component remark

For the case of color images, the method applies separately in each color component. If some of the color components are down-sampled, the mask for the down-sampled components is created in the same way as the mask of the non-down-sampled components.

H.3.2 Disjoint regions remark

If the ROI consists of disjoint parts then all parts have the same scaling value s.

H.3.3 Implementation Precision remark

This ROI coding method might in some cases create situations were the dynamic range is exceeded. This is however easily solved by simply discarding the least significant bit planes that exceeds the limit due to the down scaling operation. The effect will be that the ROI will have better quality compared to the background, even though the entire bit stream is decoded. It might however create problems when the image is coded with ROI's in a lossless mode. Discarding least significant bit-planes for the background might result that the background is not coded lossless and in the worst case not be reconstructed at all. This depends on the dynamic range available.

Annex I

Error resilience

(This annex forms an integral part of this Recommendation | International Standard)

This annex describes a method for decoding images, which have been coded using an error resilient syntax.

Many applications require the delivery of image data over different types of communication channels. Typical wireless communications channels give rise to random and burst bit errors. Internet communications are prone to loss due to traffic congestion. To improve the performance of transmitting compressed images over these error prone channels, error resilient bit stream syntax and tools are included in this specification.

The error resilience tools in this specification deal with channel errors using the following approaches: data partitioning and resynchronization, error detection and concealment, and Quality of Service (QoS) transmission based on priority. Error resilience tools are described for in each category.

Table I-1 — Error resilience tools

Type of tool	Name	Reference
Entropy coding level	code-blocks termination of the arithmetic coder for each pass reset of contexts after each coding pass selective arithmetic coding bypass	Annex C
	segmentation symbols	Annex I
Packet level	short packet format packet with resynchronization marker	Annex D

The entropy coding of the quantized coefficients is done with code-blocks. Since encoding and decoding of the code-blocks are independent, bit errors in the bit stream of a code-block will be contained to that code-block (see Annex C).

Termination of arithmetic coder is allowed after every coding pass. Also the contexts may be reset after each pass. This allows arithmetic coder continue to decode coding passes after errors (see Annex C-5).

The selective arithmetic coding bypass style puts raw bits into the bit stream without arithmetic coding. This prevents error propagation to which variable length coding is susceptible (see Annex C.6).

Short packets are achieved by removing the packet headers to the PPM or PPT marker segments (see Annex A.7.4 and Annex A.7.5). If there are errors the packet headers in the PPM or PPT marker segments can still be associated with the correct packet by using the sequence number in the SOP.

Segmentation symbols is a special symbol coded at the end of each coding pass. The correct decoding of this symbol will confirm the correctness of the decoding of this pass. This enables decoders to detect errors. A segmentation symbol is encoded with uniform context at the end of each coding pass. At the decoder, a segmentation marker “1010” should be decoded at the end of each sub-bit plane. If the segmentation symbol is not decoded correctly, then bit errors occurred for this coding pass.

Packet with resynchronization marker SOP (see Annex A.8.1) allows the spatial partitioning and resynchronization. This is placed in front of every packet in a tile with a sequence number starting at zero. It is increment with each packet. Packet ordering is described in Annex D.6.

Annex J

JP2 File format syntax

(This annex forms an integral part of this Recommendation | International Standard)

This normative annex defines an optional file format that applications may choose to use for JPEG 2000 compressed image data. While not all applications will use this format, many applications will find that this format meets their needs.

J.1 Scope

This normative annex of this Recommendation | International Standard defines an optional file format that applications may choose to use to contain JPEG 2000 compressed image data. While not all applications will use this format, many applications will find that this format meets their needs.

This annex of this Recommendation | International Standard

- specifies a binary container for both image and metadata
- specifies a mechanism to indicate the tonescale or colorspace of the image
- specifies a mechanism by which readers may recognize the existence of intellectual property rights information in the file
- specifies a mechanism by which metadata (including vendor specific information) can be included in files specified by this Recommendation | International Standard

J.2 Definitions

J.2.1 Glossary

Box: A building block defined by a unique type identifier and length. Some particular boxes may contain other boxes.

Box contents: Refers to the data wrapped within the box structure. The contents of a particular box are stored within the DBox field within the Box data structure

Box type: Specifies the kind of information that shall be stored with the box. The type of a particular box is stored within the TBox field within the Box data structure.

JP2 file: The name of file in the file format described in this specification. Structurally, a JP2 file is a contiguous sequence of boxes.

SuperBox: A Box that itself contains a contiguous sequence of boxes (and only a contiguous sequence of boxes). As the JP2 file contains only a contiguous sequence of boxes, the JP2 file is itself considered a SuperBox. When used as part of a relationship between two boxes, the term SuperBox refers to the box which directly contains the other box.

Conforming reader: An application that reads and interprets a JP2 file correctly as defined by Annex J of this Recommendation | International Standard.

Color component: A component from the codestream that functions as an input to a color transformation system. For example, a red component or a L* component would be a color component.

Auxiliary component:A component from the codestream that is used by the application outside the scope of colorspace conversion. For example, an opacity component or a depth component would be an auxiliary component.

J.2.2 Acronyms

ICC: International Color Consortium

PCS:	Profile Connection Space
UCS:	Universal Character Set
URL:	Uniform Resource Locator
UTF-8:	UCS Transformation Format 8
UUID:	Universal Unique Identifier
XML:	Extensible Markup Language

J.3 Normative References

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation |International Standard.At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation |International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below.Members of IEC and ISO maintain registers of currently valid International Standards.The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

- International Color Consortium, ICC profile format specification. ICC.1:1998–09 <http://www.color.org/ICC-1_1998-09.PDF>
- International Electrotechnical Commission. Colour management in multimedia systems: Part 2: Colour Management, Part 2–1: Default RGB colour space—sRGB. IEC 61966–2–1 199x. 9 October 1998 <http://w3.hike.te.chiba-u.ac.jp/IEC/100/ PT61966/parts/> or <http://www.sRGB.com/>.
- W3C, Extensible Markup Language (XML 1.0), Rec-xml-19980210, <http://www.w3.org/TR/REC-xml>
- IETF RFC 2279 UTF–8, A transformation format of ISO 10646. January 1998.
- ISO/IEC 11578:1996 Information technology—Open Systems Interconnection—Remote Procedure Call, <http://www.iso.ch/cate/d2229.html>

J.4 Introduction

The JPEG 2000 file format (JP2 format) provides a foundation for storing application specific data (metadata) in association with a JPEG 2000 codestream, such as that information which is required to display the image. As many applications require a similar set of information to be associated with the compressed image data, it is useful to define the format of that set of data along with the definition of the compression technology and codestream syntax.

Conceptually, the JP2 format encapsulates the JPEG 2000 codestream along with other core pieces of information about that codestream. The building-block of the JP2 format is called a box. All data is encapsulated in boxes. This Recommendation | International Standard defines several types of boxes; the definition of each specific box type defines the kinds of data that may be found within a box of that type. Some boxes will be defined to contain other boxes.

J.4.1 File identification

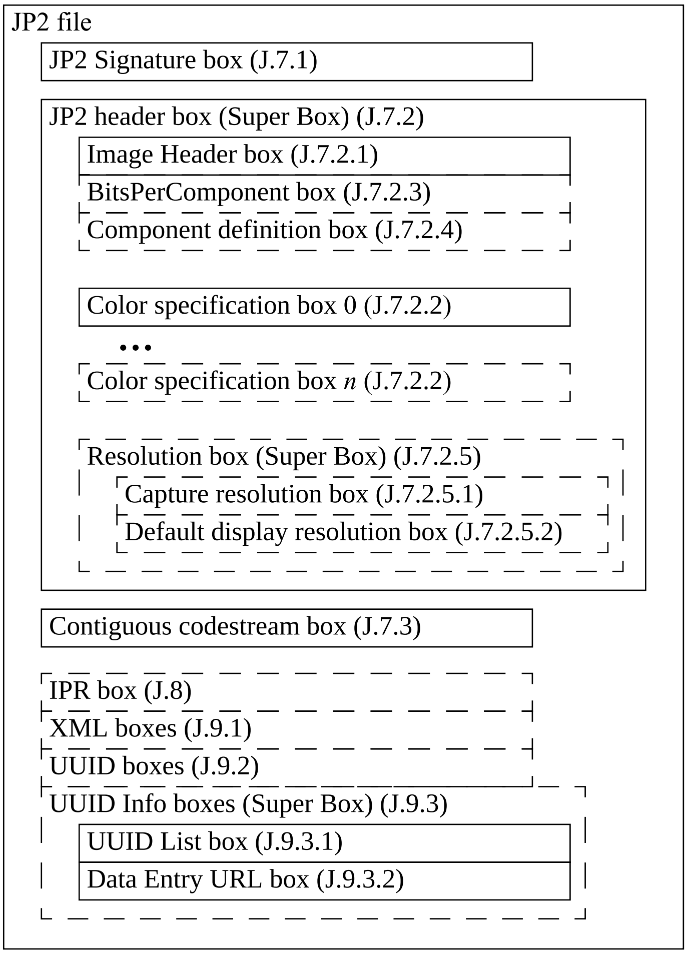
JP2 files can be identified using several mechanisms. When stored in traditional computer file systems, JP2 files should be given the file extension “.jp2” (readers shall also recognize files with the extension “.JP2”). If being transmitted as a whole (a complete, valid file is being transmitted), the internet media type “image/jpeg2000” shall be associated with the transmitted data. On Macintosh file systems, JP2 files should be given the type code ‘jp2 ’.

J.4.2 File organization

A JP2 file represents a collection of boxes. Some of those boxes are independent, and some of those boxes contain other boxes. The binary structure of a file is a contiguous sequence of boxes. The start of the first box shall be the first byte of the file, and the last byte of the last box shall be the last byte of the file.

The binary structure of a box is defined in Annex J.6.

Logically, the structure of a JP2 file is as shown in Figure J-1.


JP2 file
JP2 Signature box (J.7.1)
JP2 header box (Super Box) (J.7.2)
Image Header box (J.7.2.1)
BitsPerComponent box (J.7.2.3)
Component definition box (J.7.2.4)
Color specification box 0 (J.7.2.2)
•••
Color specification box *n* (J.7.2.2)
Resolution box (Super Box) (J.7.2.5)
Capture resolution box (J.7.2.5.1)
Default display resolution box (J.7.2.5.2)
Contiguous codestream box (J.7.3)
IPR box (J.8)
XML boxes (J.9.1)
UUID boxes (J.9.2)
UUID Info boxes (Super Box) (J.9.3)
UUID List box (J.9.3.1)
Data Entry URL box (J.9.3.2)

Boxes with dashed borders are optional in conforming JP2 files. However, an optional box may define mandatory boxes within that optional box. In that case, if the optional box exists, those mandatory boxes within the optional box shall exist. If the optional box does not exist, then the mandatory boxes within those boxes shall also not exist.

This illustration specifies only the containment relationship between the boxes in the file. A particular order of those boxes in the file is not generally implied. However, the Signature box shall be the first box in a JP2 file and the JP2 header box shall fall before the Contiguous codestream box.

Figure J-1 — Conceptual structure of a JP2 file

As shown in Figure J-1, a JP2 file contains a JP2 Signature box, JP2 header box (Super Box), and a Contiguous codestream box. A JP2 file may also contain other boxes as determined by the file writer. That JP2 header box (Super Box) contains other boxes, such as the Image Header box and one or more Color specification boxs.

J.4.3 Greyscale/Color/multi-component specification

The JP2 file format provides two methods to specify the colorspace of the image. The Enumerated method specifies the colorspace of an image by specifying a numeric value that represents a well-defined colorspace definition. In this Recommendation | International Standard, images in the sRGB colorspace and greyscale images can be defined using the enumerated method.

The JP2 file format also provides for the specification of the colorspace of an image by embedding an ICC profile in the file. That profile shall be of either the Monochrome or Three-Channel Matrix-Based class of input profiles as defined by the ICC Profile Format Specification. This allows for the specification of a wide range of greyscale and RGB class colorspaces, as well as a few other spaces that can be represented by those two profiles classes.

J.4.4 Inclusion of opacity and transparency components

The JP2 file format provides a means to indicate the presence of auxiliary components, such as opacity and transparency, to define the type of those components, and to specify the ordering of all components. When a reader opens the JP2 file, it will determine the ordering and type of each component. The application must then match the component definition and ordering from the JP2 file with the component ordering as defined by the colorspace specification. Once the file components have been mapped to the color components, the decompressed image can be processed through any needed colorspace transformations.

In many applications, components other than the color components are required. For example, many images used on web pages contain opacity information; the browser uses this information to blend the image into the background. It is thus desirable to include both the color and auxiliary components with a single codestream.

J.4.5 Metadata

One important aspect of the JP2 format is the ability to add metadata to a JP2 file. Because all data is encapsulated in boxes, and all boxes have types, the format provides a simple mechanism for a reader to extract important and understood information, while ignoring any box that contains information that is not understood by that particular reader. In this way, new boxes can be created, either through this or other Recommendations | International Standards, JURA registration, or private implementation. Also, any new box added to a JP2 file shall not change the visual appearance of the image.

J.4.6 Compliance

All conforming readers shall correctly interpret all valid JP2 files.

J.5 Greyscale/Color/multi-component specification architecture

One of the most important aspects of a file format is that it specifies the colorspace of the contained image data. In order to properly display or interpret the image data, it is essential that the colorspace of that data is properly characterized. The JP2 format provides a multi-level mechanism for characterizing the colorspace of an image. The format also provides a mechanism to specify that an image is not photographic (such as multi-spectral data).

J.5.1 Enumerated method

The simplest method for characterizing the colorspace of an image is to specify an integer code representing the colorspace in which the image is encoded. This method handles the specification of sRGB and greyscale images. Extensions to this method can be used to specify other colorspaces, including the definition of multi-component images.

For example, the image file may indicate that a particular image is encoded in the sRGB colorspace. To properly interpret and display the image, an application must natively understand the definition of the sRGB colorspace. Because an application must natively understand each specified colorspace, the complexity of this method is dependent on the exact colorspaces specified. Also, complexity of this mechanism is proportional to the number of colorspaces that are specified and required for conformance. While this method provides a high level of interoperability for images encoded using colorspaces for which correct interpretation is required for conformance, this method is very inflexible. This Recommendation | International Standard defines a specific set of colorspaces for which interpretation is required for conformance. Other colorspaces may be defined by this or other Recommendations | International Standards or by JURA registration.

J.5.2 Simple ICC profile method

An application may also specify the colorspace of an image using an selected set of simple ICC profiles. This method handles the specification of a the most commonly used RGB and greyscale class colorspaces through a low-complexity method.

An ICC profile is a standard representation of the transformation required to convert one colorspace into another colorspace. With respect to image file format, the ICC profile specification defines a type of profile that specifies how samples in a particular colorspace are converted into a standard space (the Profile Connection Space (PCS)). Depending on the original colorspace of the samples, this transformation may be either very simple or very complex.

The ICC Profile Format Specification does define a specific class of ICC profiles that are very simple. The ICC Profile Format Specification defines Monochrome Input and Three-Color Matrix-Based Input Profiles for which the transformation from the source colorspace to the PCS is limited to the application of a non-linearity curve and a 3x3 matrix. It is practical to expect all applications, including simple devices, to be able to process the image through the specified transformation. Thus all conforming applications are required to correctly interpret the colorspace of any image that specifies the colorspace using this subset of possible ICC profile types.

For this Recommendation | International Standard, the class of allowed profiles shall use the XYZ relative version of the PCS.

J.5.3 Using multiple methods

In addition to specifying these methods, this Recommendation | International Standard recognizes that applications can achieve increased interoperability or optimization by using multiple methods to specify the colorspace of an image. For example, one application may create an image that is targeted at a specific output device; the decompressed pixel values are appropriate for the direct control of the color producing apparatus in the device. In this case, the writer might use both an extended version of the enumerated colorspace method and the simple ICC profile method to specify the colorspace of the image. In this example, the device recognizes the specific colorspace (because it understands the extension to the JP2 format) and can then avoid additional processing of the image data. Applications that do not understand that enumerated colorspace can use the simple ICC profile as a means to transform the image into a colorspace appropriate for that application. By using both methods, the file can be optimized for a specific device, yet can be properly interpreted by all conforming readers.

Multiple color specification boxes are stored in the file in the order by which the file writer believes that they should be considered for use (in order of highest to lowest precedence).

A conforming JP2 file shall include at least one non-extended color specification method.

J.5.4 Interactions with the decorrelating multiple component transform

The specification of color within the JP2 file format is independent of the use of a multiple component transformation within the codestream (the CSSiz parameter of the SIZ marker segment as specified in Annex A.5.1 and Annex G). The colorspace transformations specified through the sequence of color transformation boxes shall be applied to the image samples after the reverse multiple component transformation has been applied to the decompressed samples. While the application of these decorrelating component transformations is separate, the application of an encoder-based multiple component transformation will often improve the compression of color image data.

J.6 Box definition

Physically, each object in the file is encapsulated within a binary structure called a box. That binary structure is as follows:

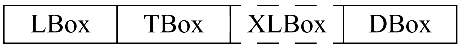


Figure J-2 — Organization of a Box

LBox: Box Length. This field specifies the length of the box, stored as a 4-byte unsigned integer. This value includes all of the fields of the box, including the length and type. If the value of this field is 1, then the XLBox field shall exist and the value of that field shall be the actual length of the box. If the value of this field is 0, then the length of the box was not know when the LBox field was written. In this case, this box contains all data up to the end of this box's SuperBox. The values 2–7 are reserved for other use.

TBox: Box Type. This field specifies the type of data found in the DBox field. The value of this field is encoded as a four-character ASCII string with no terminating null. For example, the type 'jp2h' specifies a box that contains the header object for a JP2 file. This value can also be considered to be a 32-bit unsigned integer. For all box types defined within this Recommendation | International Standard, box types will be specified both as 4-character strings and as 32-bit integers in hexadecimal.

XLBox: Box Extended Length. This field specifies the actual length of the box if the value of the LBox field is 1. This field is stored as an 8-byte unsigned integer. The value includes all of the fields of the box, including the LBox, TBox and XLBox fields.

DBox: Box Data. This field contains the data for the portion of the object contained within this box. The format of that data is dependent on the box type and will be defined individually for each type.

Table J-1 — Binary structure of a box

Field name	Size (bits)	Value
LBox	32	0, 1, 8—(2^{32}–1)
TBox	32	
XLBox	LBox=1, 64 LBox≠1, 0	16—(2^{64}–1)
DBox	Varies	

For example, consider the following illustration of a sequence of boxes, including one box that contains other boxes:

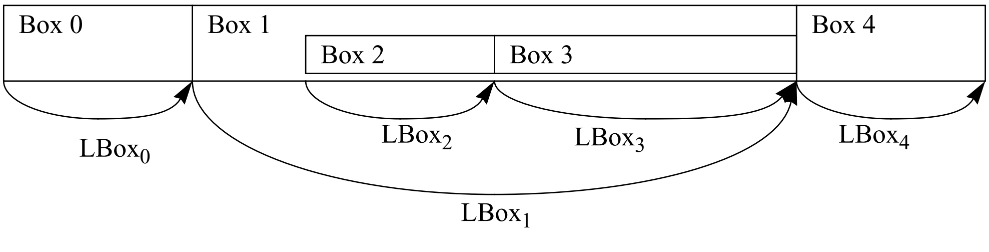


Figure J-3 — Illustration of box lengths

As shown in Figure J-3, the length of each box includes any boxes contained within that box. For example, the length of Box 1 includes the length of Boxes 2 and 3, in addition to the LBox and TBox fields for Box 1 itself. In this case, if the type of Box 1 was not understood by a reader, it would not recognize the existence of boxes 2 and 3 because they would be completely skipped by jumping the length of box 2 from the beginning of box 2.

The following table lists all boxes defined by this Recommendation | International Standard:

Table J-2 — Boxes defined within this Recommendation | International Standard

Box name	Type	SuperBox	Mandatory?	Notes
JP2 Signature box	'jP2\032' (X'6A50321A')	No	Mandatory	This box uniquely identifies the file as a JP2 file.
JP2 Header box	'jp2h' (X'6A703268')	Yes	Mandatory	This box contains header information about the file.
Image Header box	'ihdr' (X'69686472')	No	Mandatory	This box contains header information about the image itself.
BitsPerComponent box	'bpcc' (X'62706363')	No	Optional	This box specifies the bit depth of the components in the file in cases where the bit depth is not constant across all components.
Component Definition box	'cdef' (X'63646566')	No	Optional	This box specifies the type and ordering of the components within the codestream.
Color specification	'colr' (X'636F6C72')	No	Required	This box specifies the colorspace of the image.
Resolution box	'res ' (X'72657320')	Yes	Optional	This box specifies the resolution of the image.
Capture resolution box	'resc' (X'72657363')	No	Optional	This box specifies the resolution at which the image was captured.
Default Display resolution box	'resd' (X'72657364')	No	Optional	This box specifies the default resolution at which the image should be displayed.
Contiguous codestream box	'jp2c' (X'6A703263')	No	Required	This box contains the codestream as defined by Annex A of this Recommendation | International Standard
Intellectual Property box	'jp2i' (X'6A703269')	No	Optional	This box contains intellectual property information about the image.
XML box	'xml ' (X'786D6C20')	No	Optional	This box provides a tool by which vendors can add XML formatted information to a JP2 file.
UUID box	'uuid' (X '75756964')	No	Optional	This box provides a tool by which vendors can add additional data to a file without risking conflict with other vendors.

Table J-2 — Boxes defined within this Recommendation | International Standard

Box name	Type	SuperBox	Mandatory?	Notes
UUID Info box	‘uinf’ (X‘75696E66’)	Yes	Optional	This box provides a tool by which a vendor may provide access to additional information associated with a UUID
UUID list box	‘ulst’ (X‘75637374’)	No	Optional	This box specifies a list of UUID’s.
URI box	‘url ’ (X‘75726C20’)	No	Optional	This box specifies a URI.

J.7 Defined boxes

The following boxes shall properly be interpreted by all conforming readers.

J.7.1 JP2 Signature box

The JP2 signature box identifies this file as a JP2 file, as well as provides a small amount of information which can help determine the probable validity of the rest of the file. The JP2 signature box shall be the first box in the file, and all files shall contain one and only one JP2 signature box.

The type of the JP2 signature box shall be ‘jP2\032’ (X‘6A50321A’). The length of this box shall be 12 bytes. The contents of this box shall be the 4-byte character string ‘<CR><LF><X’87’><LF>’ (X‘0D0A870A’). For file verification purposes, this box can be considered a fixed-length 12-byte string which shall have the value:
X‘0000 000C 6A50 321A 0D0A 870A’.

The combination of the particular type and contents for this box enable an application to detect a common set of file transmission errors. The CR-LF sequence in the contents catches bad file transfers that alter newline sequences. The control-Z character in the type stops file display under MS-DOS. The final linefeed checks for the inverse of the CR-LF translation problem. The third character of the box contents has its high-bit set to catch bad file transfers that clear bit 7.

J.7.2 JP2 header box (Super Box)

The JP2 header box contains generic information about the file, such as number of samples, colorspace, and resolution. This box is a SuperBox.

Within a JP2 file (considered as a Super Box), there shall be one and only one JP2 header box. The JP2 header box may be located anywhere within the file after the JP2 signature box but before the contiguous codestream box. It also must be at the same level as the JP2 signature box (it shall not be inside any other Super Box within the file).

The type of the JP2 header box shall be ‘jp2h’ (X‘6A703268’).

This box contains several boxes, some of which will be defined within this Recommendation | International Standard, and some of which will be defined in other standards. Those boxes contained within the JP2 header box that are defined within this Recommendation | International Standard are as follows:

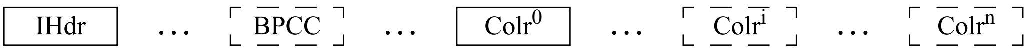


Figure J-4 — Organization of the contents of a JP2 header box

IHdr: This field specifies information about the image, such as its height, width and resolution. This structure is an Image Header box as specified in Annex J.7.2.1. This box shall be the first box in the JP2 header box.

BPCC: This box specifies the bit depth of each component in the codestream after decompression. The value of this structure is a BPCC box as specified in Annex J.7.2.3. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

Colr^{i}: These fields specify the colorspace of the decompressed image. The value of each of these structures is a Color specification box as specified in Annex J.7.2.2. There shall be at least one Color specification box within the JP2 header box. The use of multiple color specification boxes provides the ability for a decoder to be given multiple optimization or compatibility options for color processing. The sequence of the color specification boxes within the header box represents the precedence associated with each colorspace specification, with higher precedence representations coming before lower precedence ones. These boxes may be found anywhere in the JP2 header box provided that they come after the Image Header box, and are in order based on precedence.

J.7.2.1 Image Header box

This box contains generic information about the image, such as the image size and number of components. The JP2 header box shall contain one instance of this box at the beginning of its DBox field. Instances of this box in other places in the file shall be ignored. The length of the Image Header box shall be 14 bytes.

The type of the Image Header box shall be ‘ihdr’ (X‘69686472’) and contents of the box shall have the following format:

VERS	NC	HEIGHT	WIDTH	BPC	C

Figure J-5 — Organization of the contents of an Image Header box

VERS: Version. This parameter defines the version number of this JP2 specification for which the file complies. The parameter is defined as a 2-byte unsigned integer with the most significant byte containing the major version number (currently defined as 1) and the least significant byte containing a minor revision number (currently defined as 0).

The value of this field is X‘0100.’

A major version number increment (if there ever is one) represents an incompatible change in JP2 files. Decoders should give up if the encounter an unrecognized major version number. Minor version number increments represent backward compatible changes. Decoders should continue to process JP2 files even if the minor version number is unrecognized.

NC: Number of components. This parameter specifies the number of components in the image and is stored as a 2-byte unsigned integer.

HEIGHT: Image height. The value of this parameter indicates the number of lines in the tallest component of the image and is stored as a 4-byte unsigned integer.

WIDTH: Image width. The value of this parameter indicates the number of samples per line in the widest component of the image and is stored as a 4-byte unsigned integer.

BPC: Bits per component. This parameter specifies the bit depth of the components in the image and is stored as a 1-byte ones-complement integer.

If the bit depth is the same for all components, then this field specifies the actual bit depth. If the components vary in bit depth, then the value of this field shall be zero and the JP2 header box shall also contain a BitsPerComponent box defining the bit depth of each component (as defined in Annex J.7.2.3).

The low 7-bits of the value indicate the bit depth of the components. The high-bit indicates whether the components are signed or unsigned. If the high-bit is 1, then the components contain signed values. If the high-bit is 0, then the components contain unsigned values.

C: Compression type. This parameter specifies the compression algorithm used to compress the image data. The value of this field shall be 7. It is encoded as a 1-byte unsigned integer. If the value of this field is not 7, then this file is not a conforming JP2 file.

Table J-3 — Format of the contents of the Image Header box

Field name	Size (bits)	Value
VERS	16	X‘0100’
NC	16	1—(2^{16}–1)
HEIGHT	32	1—(2^{32}–1)
WIDTH	32	1—(2^{32}–1)
BPC	8	-127—127
C	8	7

J.7.2.2 Color specification box

Each Color specification box defines one method by which an application can interpret the colorspace of the decompressed image data. A JP2 file may contain multiple Color specification boxes, specifying different methods for achieving “equivalent” results. Note that this color specification is to be applied to the image data after it has been decompressed and after any reverse decorrelating component transform has been applied to the data. The type of a Color Specification box shall be ‘colr’ (X‘636F6C72’). The contents of a Color specification box is as follows:

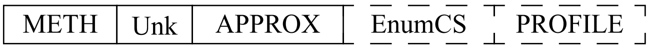


Figure J-6 — Organization of the contents of a Color specification box

METH: Specification method. This field specifies the method used by this Color specification box to define the colorspace of the decompressed image. This field is encoded as a 1-byte unsigned integer. Legal values of this field are as follows:

Table J-4 — Legal METH values

Value	Meaning
1	**Enumerated colorspace**. This colorspace specification box contains the enumerated value of the colorspace of this image. The enumerated value is found in the EnumCS field in this box. If the value of the METH field is 1, then the EnumCS shall exist in this box immediately following the APPROX field, and the EnumCS field shall be the last field in this box
2	**Simple ICC profile**.This color specification box contains a simple ICC profile in the PROFILE field. This profile specifies the transformation needed to convert the decompressed image data into the PCS. If the value of METH is 2, then the ICC profile shall conform to the definition of either a Monochrome Input Profile or a Three-Component Matrix-Based Input Profile as defined in the ICC profile specification, version 2.2.0. In addition, the value of the Profile Connection Space field in the profile header in the embedded profile shall be ‘XYZ ’ (X‘58595A20’) indicating that the output colorspace of the simple profile is XYZ data. If the value of METH is 2, then the PROFILE field shall immediately follow the APPROX field and the PROFILE field shall be the last field in the box.

Table J-4 — Legal METH values

Value	Meaning
other values	Reserved for other ISO use. If the value of METH is not 1 or 2, there may be fields in this box following the APPROX field. Those fields shall be ignored.

Unk: Colorspace Unknown. This field specifies if the actual colorspace of the image data is known. This field is encoded as a 1-byte unsigned integer. Legal values for this field are 1, if the colorspace of the image is known and correctly specified by this color specification box, or 0, if the colorspace of the image is not known. A value of 0 will be used in cases such as the transcoding of legacy images where the actual colorspace of the image data is not known. In those cases, while the colorspace interpretation methods specified in the file may not accurately reproduce the image with respect to some original, the image should be treated as if the methods do accurately reproduce the image. Values other than 0 and 1 are reserved for other use.

APPROX: Colorspace approximation. This field specifies the extent to which this color specification method approximates the "correct" definition of the colorspace. This field is encoded as a 1-byte unsigned integer with values from 0–5. A value of zero indicates that the degree that this definition approximates the "correct definition is not known. Values between 1 and 5 indicate a respective degree of approximation, where a value of 1 indicates that this definition is a poor approximation of the correct definition, and a value of 5 indicates that this definition is exactly the correct definition.

EnumCS: Enumerated colorspace. This field specifies the colorspace of the image using integer codes. To correctly interpret the color of an image using an enumerated colorspace, the application must know the definition of that colorspace internally. This field contains a 4-byte unsigned integer value indicating the colorspace of the image. If the value of the METH field is 2, then the EnumCS field shall not exist. The following colorspace codes are defined by this Recommendation | International Standard:

Table J-5 — Legal EnumCS values

Value	Meaning
16	sRGB as defined by IEC 61966–2–1
17	greyscale: A single-channel greyscale space. Images encoded in this space consist of Y values as defined by IEC 61966–2–1 (sRGB) Equation 5.
other values	Reserved for other ISO uses

PROFILE:ICC profile. This field contains a valid ICC profile, as specified by the ICC Profile Format Specification, which specifies the transformation of the decompressed image data into the PCS. This field shall not exist if the value of the METH field is 1. If the value of the METH field is 2, then the ICC profile shall conform to the Monochrome Input Profile class or the Three-Component Matrix-Based Input Profile class as defined in the ICC profile specification.

Table J-6 — Format of the contents of the Colr box

Field name	Size (bits)	Value
METH	8	1—2
Unk	8	0—1
APPROX	8	0—5

Table J-6 — Format of the contents of the Colr box

Field name	Size (bits)	Value
EnumCS	0 if METH=1 8 if METH=2	no value 1—255
PROFILE	Varies	Varies

J.7.2.3 BitsPerComponent box

The BitsPerComponent box specifies the bit depth of each component. If the bit depth is constant across all components in the codestream, then this box shall not be found. Otherwise, this box specifies the bit depth of each component. The order of bit depth values in this box is the actual order those components are enumerated within the codestream. The exact location of this box within the JP2 header box may vary provided that it follows the Image Header box.

The type of the BitsPerComponent Box shall be ‘bpcc’ (X‘62706363’). The contents of this box shall be as follows:

$$\boxed{\mathrm{BPC}^{0}}\quad \ldots \quad \boxed{\mathrm{BPC}^{i}}\quad \ldots \quad \boxed{\mathrm{BPC}^{NC-1}}$$

Figure J-7 — Organization of the contents of a BitsPerComponent box

BPC^{i}: Bits per component. This parameter specifies the bit depth of component *i*, encoded as a 1-byte ones-complement integer. The ordering of the components within the BitsPerComponent Box shall be the same as the ordering of the components within the codestream. The number of BPC^{i} fields shall be the same as the value of the NC field from the Image Header box.

The low 7-bits of the value indicate the bit depth of this component. The high-bit indicates whether the component is signed or unsigned. If the high-bit is 1, then the component contains signed values. If the high-bit is 0, then the component contains unsigned values.

Table J-7 — Format of the contents of the BitsPerComponent box

Field name	Size (bits)	Value
BPC^{i}	8	-127— -1, 1—127

J.7.2.4 Component definition box

The component definition box specifies the meaning of the data in each component in the codestream. The exact location of this box within the JP2 header box may vary provided that it follows the Image Header box.

This box contains an array of component descriptions. For each description, three values are specified: the number of the component described by that association, the type of that component, and the association of that component with particular colors. This box may specify multiple descriptions for a single component; however, the type value in each description for the same component shall be the same in all descriptions.

If the codestream contains only color components and those components are ordered in the same order as the associated colors (for example, an RGB images with three components in the order R, G, then B), then this box shall not exist. However, if there are any auxiliary components or the components are not in the same order as the color numbers, then the Component definition box shall be found within the JP2 header box with a complete list of component definitions.

If a multiple component transform is specified within the codestream, the component ordering box shall specify the existence of red, green and blue colors as components 0, 1 and 2 in the codestream, respectively.

The type of the Component definition box shall be ‘cdef’ (X‘63646566’). The contents of this box shall be as follows:

N	Cn^0	Typ^0	Asoc^0	. . .	Cn^{N-1}	Typ^{N-1}	Asoc^{N-1}

Figure J-8 — Organization of the contents of a Component definition box

N: Number of component descriptions. This field specifies the number of component descriptions in this box. This field is encoded as a two-byte unsigned integer.

$\mathbf{Cn}^{\mathbf{i}}$: Component number. This field specifies the number of the component for this description. The value of this field represents the number of the component as defined within the codestream. This field is encoded as a two-byte unsigned integer.

$\mathbf{Typ}^{\mathbf{i}}$: Component type. This field specifies the type of the component for this description. The value of this field represents the type of data contained within the component. This field is encoded as a two-byte unsigned integer. Legal values of this field are as follows:

Table J-8 — $\mathbf{Typ}^{\mathbf{i}}$ field values

Value	Meaning
0	This component is the color component for the associated color
1	Opacity. A sample value of 0 indicates that the sample is 100% transparent, and the maximum value of the component (related to the bit depth of the component) indicates a 100% opaque sample.
2	Premultiplied opacity. An opacity component as specified above, except that the value of the opacity component has been multiplied into the color components for which this component is associated. Premultiplication is defined as follows: $$S_p = S \times \frac{\alpha}{\alpha_{max}} \qquad \text{J.1}$$ where S is the original sample, S_p is the premultiplied sample (the sample stored in the image, α is the value of the opacity component, and α_{max} is the maximum value of the opacity component as defined by the bit depth of the opacity component.
3—(2^{16}–2)	Reserved for ISO use
2^{16}–1	The type of this component is not specified

$\mathbf{Asoc}^{\mathbf{i}}$: Component association. This field specifies the number of the color for which this component is directly associated (or a special value to indicate the whole image or the lack of an association). For example, if this component is an opacity blending component for the red component in an RGB colorspace, this field would specify the number of the color red. Table J-9 specifies legal association values. Table J-10 specifies legal color numbers. This field is encoded as a two-byte unsigned integer.

Table J-9 — $\mathbf{Asoc}^{\mathbf{i}}$ field values

Value	Meaning
0	This component is associated as the image as a whole (for example, a component independent opacity blending channel

Table J-9 — Asoc^i field values

Value	Meaning
1—(2^{16}–2)	This component is associated with the a particular color as indicated by this value. This value is used to associate a particular component with a particular aspect of the specification of the colorspace of this image. For example, indicating that a component is associated with the red component of an RGB image allows the reader to associate that decoded component with the Red input to an ICC profile contained within a color specification box. Color indicators are specified in Table J-10
2^{16}–1	This component is not associated with any particular color

Table J-10 — Colors indicated by the Asoc^i field

Class of colorspace	Color indicated by the following value of the Asoc^i field			
	1	2	3	4
RGB	R	G	B	
Greyscale	Y			
The following colorspace classes are listed for future reference, as well as to aid in understanding of the use of the Asoc^i field				
XYZ	X	Y	Z	
Lab	L	a	b	
Luv	L	u	v	
YC_bC_r	Y	C_b	C_r	
Yxy	Y	x	y	
HSV	H	S	V	
HLS	H	L	S	
CMYK	C	M	Y	K
CMY	C	M	Y	
Jab	J	a	b	
n color colorspaces	1	2	3	4

In this box, component numbers refer to the number of that particular component within the codestream. Color numbers specify how that component shall be interpreted based on the specification of the colorspace of the image.

For example, the green color in an RGB image is specified by a {Cn, Typ, Asoc} value of {*i*, 0, 2}, where *i* is the number of that component in the codestream (either directly or as generated by applying the reverse multiple component transform). Applications that are only concerned with extracting the color components can treat the Typ/Asoc field pair as a four-byte value where the combined value maps directly to the color numbers (as the Typ field for a color component shall be 0).

In another example, the codestream may contain a component *i* that specifies opacity blending data for the red and green components, and a component *j* that specifies opacity blending data for the blue component. In that file, the following {Cn, Typ, Asoc} tuples would be found in the Component definition box: {*i*, 1, 1}, {*i*, 1, 2} and {*j*, 1, 3}.

There shall not be more than component in a JP2 file with a Typ^i value of 0 with the same value of the Asoc^i field. For example a JP2 file in an RGB colorspace shall only contain one green component.

Table J-11 — Component definition & ordering data structure values

Parameter	Size (bits)	Value
N	16	0—(2^{16}–1)
Cn^i	16	0—(2^{16}–1)
Typ^i	16	0—(2^{16}–1)
Asoc^i	16	0—(2^{16}–1)

J.7.2.5 Resolution box (Super Box)

This box specifies the capture and default display resolution of this image. If this box exists, it shall contain either a capture display resolution box, or a default display resolution box, or both.

The type of a Resolution box shall be ‘res ’ (X‘72657320’). The contents of the resolution box are as follows:

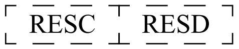


Figure J-9 — Organization of the contents of Resolution box

RESC: Capture resolution box. This box specifies the resolution at which this image was captured. The format of the RESC box is specified in Annex J.7.2.5.1.

RESD: Default display resolution box. This box specifies the default resolution at which this image should be displayed. The format of a RESD box is specified in Annex J.7.2.5.2

J.7.2.5.1 Capture resolution box

This box specifies the resolution at which this image was captured. This specifies the resolution at which the source was digitized to create the image samples specified by the codestream. For example, this may specify the resolution of the flatbed scanner that captured a page from a book. The capture resolution could also specify the resolution of an aerial digital camera or satellite camera.

The vertical and horizontal capture resolutions are calculated using the following two equations, respectively:

$$VRc = \frac{VRcN}{VRcD} \times 10^{VRcE} \qquad \text{J.2}$$

$$HRc = \frac{HRcN}{HRcD} \times 10^{HRcE} \qquad \text{J.3}$$

The values *VRc* and *HRc* are always in samples/meter. If an application requires the resolution in another unit, then that application must apply the appropriate conversion.

The type of a Capture resolution box shall be ‘resc’ (X‘72657363’). The contents of the Capture resolution box are as follows:

VRcN	VRcD	HRcN	HRcD	VRcE	HRcE

Figure J-10 — Organization of the contents of Resolution box

VRcN: Vertical Capture resolution numerator. This parameter specifies the *VRcN* value in Equation J.1, which is used to calculate the vertical capture resolution. This parameter is encoded as an 16-bit unsigned integer.

VRcD: Vertical Capture resolution denominator. This parameter specifies the *VRcD* value in Equation J.1, which is used to calculate the vertical capture resolution. This parameter is encoded as an 16-bit unsigned integer.

HRcN: Horizontal Capture resolution numerator. This parameter specifies the *HRcN* value in Equation J.2, which is used to calculate the horizontal capture resolution. This parameter is encoded as an 16-bit unsigned integer.

HRcD: Horizontal Capture resolution denominator. This parameter specifies the *HRcD* value in Equation J.2, which is used to calculate the horizontal capture resolution. This parameter is encoded as an 16-bit unsigned integer.

VRcE: Vertical Capture resolution exponent. This parameter specifies the *VRcE* value in Equation J.1, which is used to calculate the vertical capture resolution. This parameter is encoded as an 8-bit signed integer.

HRcE: Horizontal Capture resolution exponent. This parameter specifies the *HRcE* value in Equation J.2, which is used to calculate the horizontal capture resolution. This parameter is encoded as an 8-bit signed integer.

Table J-12 — Format of the contents of the Capture resolution box

Field name	Size (bits)	Value
VRcN	16	1—(2^{16}–1)
VRcD	16	1—(2^{16}–1)
HRcN	16	1—(2^{16}–1)
HRcD	16	1—(2^{16}–1)
VRcE	8	-128—127
HRcE	8	-128—127

J.7.2.5.2 Default display resolution box

This box specifies the default resolution at which this image should be displayed. This specifies a default resolution at which the image should be displayed. For example, this may be used to determine the size of the image on a page when the image is placed in a page-layout program. Note, however, that this value is only a default. Each application must determine an appropriate display size for that application.

The vertical and horizontal display resolutions are calculated using the following two equations, respectively:

$$VRd = \frac{VRdN}{VRdD} \times 10^{VRdE} \qquad \text{J.4}$$

$$HRd = \frac{HRdN}{HRdD} \times 10^{HRdE} \qquad \text{J.5}$$

The values *VRd* and *HRd* are always in samples/meter. If an application requires the resolution in another unit, then that application must apply the appropriate conversion.

The type of a Default display resolution box shall be ‘resd’ (X‘72657364’). The contents of the Default display resolution box are as follows:

VRdN	VRdD	HRdN	HRdD	VRdE	HRdE

Figure J-11 — Organization of the contents of Resolution box

VRdN: Vertical Display resolution numerator. This parameter specifies the *VRdN* value in Equation J.3, which is used to calculate the vertical display resolution. This parameter is encoded as an 16-bit unsigned integer.

VRdD: Vertical Display resolution denominator. This parameter specifies the *VRdD* value in Equation J.3, which is used to calculate the vertical display resolution. This parameter is encoded as an 16-bit unsigned integer.

HRdN: Horizontal Display resolution numerator. This parameter specifies the *HRdN* value in Equation J.4, which is used to calculate the horizontal display resolution. This parameter is encoded as an 16-bit unsigned integer.

HRdD: Horizontal Display resolution denominator. This parameter specifies the *HRdD* value in Equation J.4, which is used to calculate the horizontal display resolution. This parameter is encoded as an 16-bit unsigned integer.

VRdE: Vertical Display resolution exponent. This parameter specifies the *VRdE* value in Equation J.3, which is used to calculate the vertical display resolution. This parameter is encoded as an 8-bit signed integer.

HRdE: Horizontal Display resolution exponent. This parameter specifies the *HRdE* value in Equation J.4, which is used to calculate the horizontal display resolution. This parameter is encoded as an 8-bit signed integer.

Table J-13 — Format of the contents of the Default display resolution box

Field name	Size (bits)	Value
VRdN	16	1—(2^{16}–1)
VRdD	16	1—(2^{16}–1)
HRdN	16	1—(2^{16}–1)
HRdD	16	1—(2^{16}–1)
VRdE	8	-128—127
HRdE	8	-128—127

J.7.3 Contiguous codestream box

The Contiguous codestream box contains a valid and complete JPEG 2000 codestream, as defined in Annex A of this Recommendation | International Standard. Exactly one contiguous codestream box shall be found in the file.

The type of the contiguous codestream box shall be ‘jp2c’ (X‘6A703263’). The contents of the box shall be as follows:

Code

Figure J-12 — Organization of the contents of the Contiguous codestream box

Code: This field contains a valid and complete JPEG 2000 codestream as specified by Annex A of this Recommendation | International Standard.

Table J-14 — Format of the contents of the Contiguous codestream box

Field name	Size (bits)	Value
Code	Varies	Varies

J.8 Adding intellectual property rights information in JP2

This Recommendation | International Standard specifies a box type for a box which is devoted to carrying intellectual property rights information within a JP2 file. Inclusion of this information in a JP2 file is optional for conforming files. The definition of the format of the contents of this box is reserved for ISO. However, the type of this box is defined here as a means to allow applications to recognize the existence of IPR information.

The type of the Intellectual Property Box shall be ‘jp2i’ (X‘6A703269’).

J.9 Adding vendor specific information to the JP2 file format

The following boxes provide a set of tools by which applications can add vendor specific information the JP2 file format. All of the following boxes are optional in conforming files and may be ignored by conforming readers.

J.9.1 XML boxes

An XML box contains vendor specific data other than that data defined within this Recommendation | International Standard. There may be multiple XML boxes within the file, and those boxes may be found anywhere in the file except before the JP2 signature box.

The type of an XML box is ‘xml ’ (X’786D6C20’). The contents of the box shall be as follows:

Figure J-13 — Organization of the contents of the Contiguous codestream box

XML: This field shall be valid XML as defined by REC-xml-19980210.

The existence of any XML boxes is optional for conforming files. Also, any XML box shall not contain any information necessary for decoding the image to the extent that is defined within this part of this Recommendation | International Standard, and the correct interpretation of the data in any XML box shall not change the visual appearance of the image. All readers may ignore any XML box in the file.

J.9.2 UUID boxes

A UUID box contains vendor specific data other than that data defined within this Recommendation | International Standard. There may be multiple UUID boxes within the file, and those boxes may be found anywhere in the file except before the JP2 signature box.

The type of a UUID box shall be ‘uuid’ (X‘75756964’). The contents of the box shall be as follows:

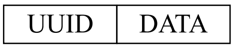


Figure J-14 — Organization of the contents of the Contiguous codestream box

UUID: This field contains a 16-byte UUID as specified by ISO/IEC 11578:1996. The value of this UUID specifies the format of the vendor specific data stored in the DATA field and the interpretation of that data.

DATA: This field contains the vendor specific data. The format of this data is defined outside of the scope of this standard, but is indicated by the value of the UUID field.

Table J-15 — Format of the contents of a UUID box

Field name	Size (bits)	Value
UUID	128	Varies
DATA	Varies	Varies

The existence of any UUID boxes is optional for conforming files. Also, any UUID box shall not contain any information necessary for decoding the image to the extent that is defined within this part of this Recommendation | International Standard, and the interpretation of the data in any UUID box shall not change the visual appearance of the image. All readers may ignore any UUID box.

J.9.3 UUID Info boxes (Super Box)

While it is useful to allow vendors to extend JP2 files by adding binary data using UUID boxes, it is also useful to provide information in a standard form which can be used by non-extended applications to get more information about the extensions in the file. This information is contained in UUID Info boxes. A JP2 file may contain zero or more UUID Info boxes. These boxes may be found anywhere in the top level of the file (the SuperBox of a UUID Info box shall be the JP2 file itself) except before the signature box.

Note that these boxes, if present, may not provide a complete index for the UUID's in the file, may reference UUID's not used in the file, and possibly may provide multiple references for the same UUID.

The type of a UUID Info box shall be 'uinf' (X'75696E66'). The contents of a UUID Info box are as follows:

UList	DE

Figure J-15 — Organization of the contents of a UUID Info box

UList: UUID List box. This box contains a list of UUID's for which this UUID Info box specifies a link to more information. The format of the UUID List box is specified in Annex J.9.3.1.

DE: Data Entry URL box. This box contains either URL. An application can acquire more information about the UUID's contained in the UUID list box. The format of a Data Entry URL box is specified in Annex J.9.3.2

J.9.3.1 UUID List box

This box contains a list of UUID's. The type of a UUID List box shall be 'ulst' (X'75637374'). The contents of a UUID list box shall be as follows:

NU	$UUID^{0}$	...	$UUID^{NU-1}$

Figure J-16 — Organization of the contents of a UUID Info box

NU: Number of UUID's. This field specifies the number of UUID's found in this UUID List box. This field is encoded as a 16-bit unsigned integer.

UUID^i:UUID. This field specifies one UUID, as specified in ISO/IEC 11578:1996, which shall be associated with the URI contained in the URI box within the same UUID Info box. The number of UUID^i fields shall be the same as the value of the NU field. The value of this field shall be a 16-byte UUID.

Table J-16 — UUID List box contents data structure values

Parameter	Size (bits)	Value
NU	16	0—(2^{16}–1)
UUID^i	128	0—(2^{128}–1)

J.9.3.2 Data Entry URL box

This box contains a URL which can use used by an application to acquire more information about the associated vendor specific extensions. The format of the data acquired through the use of this URL is not defined in this Recommendation | International Standard. The URL type should be of a service which delivers a file (e.g. URL's of type file, http, ftp, etc.), which ideally also permits random access. Relative URL's are permissible and are relative to the file containing this data reference.

The type of a Data Entry URL box shall be 'url ' (X'75726C20'). The contents of a Data Entry URL box shall be as follows:

Figure J-17 — Organization of the contents of a URI box

VERS: Version number. This field specifies the version number of the format of this box. The value of this field shall be 0.

FLAG: Flags. This field is reserved for other use to flag particular attributes of this box. The value of this field shall be 0.

URL: URL. This field specifies the URL of the additional information associated with the UUID's contained in the UUID List box within the same UUID Info SuperBox. The URL is encoded as a null terminated string of UTF-8 characters

Table J-17 — URI box contents data structure values

Parameter	Size (bits)	Value
VERS	8	0
FLAG	24	0
URL	varies	varies

J.10 Dealing with unknown boxes

A valid codestream may contain boxes not known to applications based solely on this Recommendation | International Standard. If an application finds a box that it does not understand, it shall skip and ignore that box.

Annex K

Descriptions of encoding and implementation

This Annex is an informative description for encoding and implementation.

K.1 Adaptive Entropy Software-Conventions Decoder

This annex provides some alternative flowcharts for a version of the adaptive entropy decoder. This alternative version may be more efficient when implemented in software, as it has fewer operations along the fast path. This annex is strictly informative.

The alternative version is obtained by making the following substitutions.

Replace the flowchart in Figure B-20 with the flowchart in Figure K-1.

Replace the flowchart in Figure B-15 with the flowchart in Figure K-2.

Replace the flowchart in Figure B-19 with the flowchart in Figure K-3.

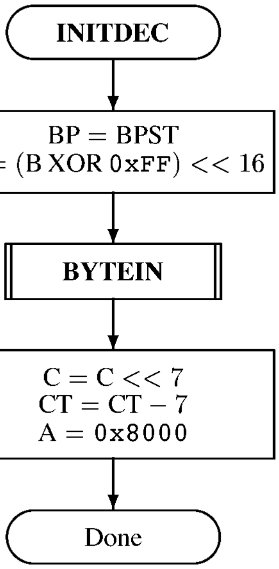


Figure K-1 — Initialisation of the software-conventions decoder

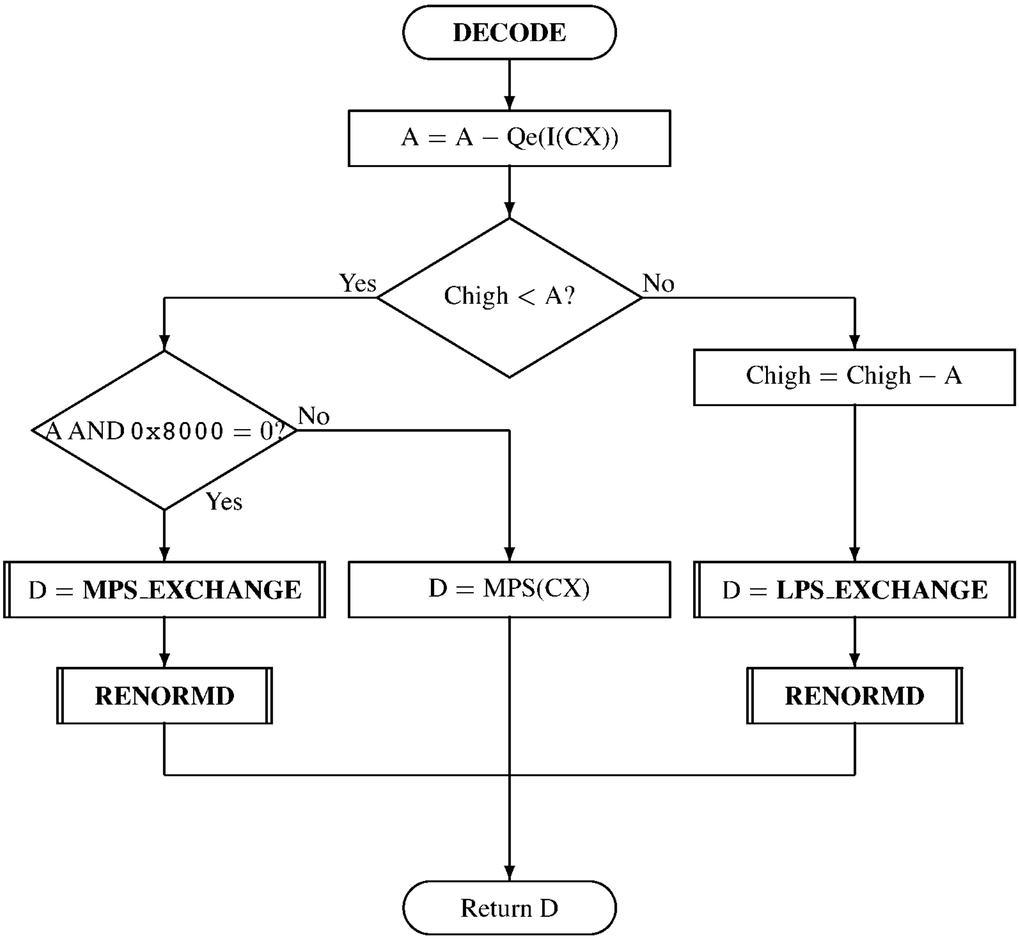


Figure K-2 — Decoding an MPS or an LPS in the software-conventions decoder

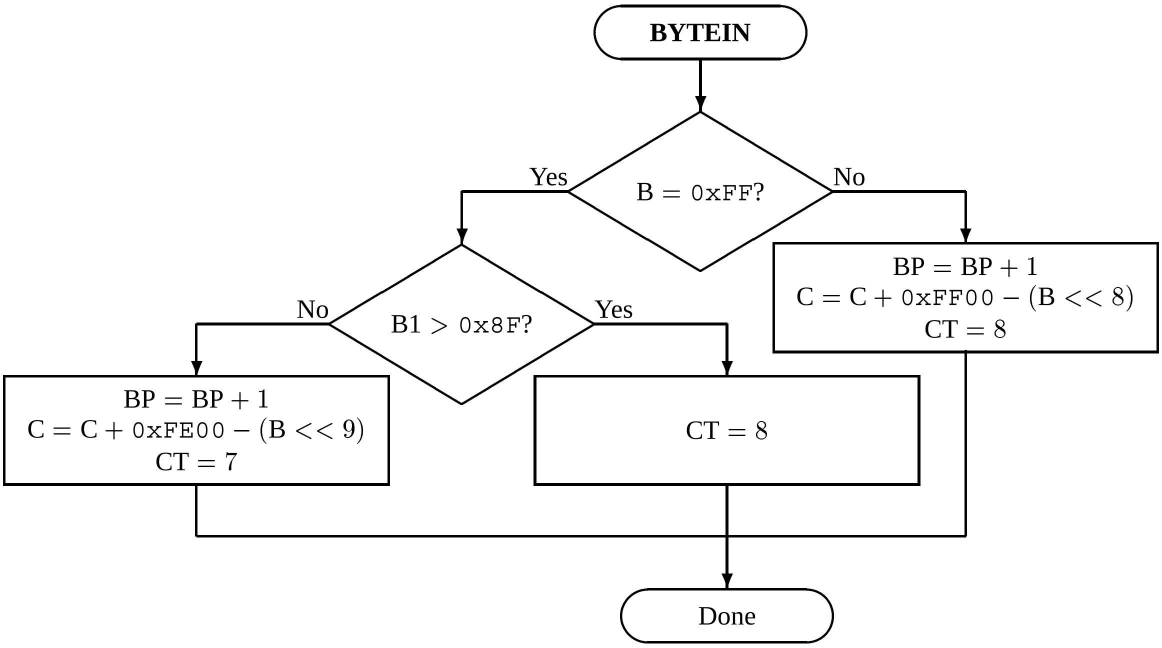


Figure K-3 — Inserting a new byte into the C register in the software-conventions decoder

Annex L

Examples and guidelines

This Annex includes a number of examples intended to indicate how the encoding process works, and how the resulting data stream should be output. This Annex is entirely informational.

L.1 Visual Frequency Weighting

The human visual system plays an important role in the perceived image quality of compressed images. It is therefore desirable to allow system designers and users to take advantage of the current knowledge of visual perception, e.g., to utilize models of the visual system's varying sensitivity to spatial frequencies, as measured in the contrast sensitivity function (CSF). Since the CSF weight is determined by the visual frequency of the transform coefficient, there will be one CSF weight per sub-band in the wavelet transform. The design of the CSF weights is an encoder issue and depends on the specific viewing condition under which the decoded image is to be viewed. Please refer to [31][32][33][34] for more details of the design of the CSF weights.

In many cases, only one set of CSF weights is chosen and applied according to the viewing condition. This application of visual frequency weighting is referred to as the fixed visual weighting. In the case of embedded coders, as the coding bit stream may be truncated later, the viewing conditions at different stages of embedding may be very different. At low bit rates, the quality of the compressed image is poor and the detailed features of the image are not available. The image is usually viewed at a relatively large distance and the observers are more interested in the global features. As more and more bits are received, the image quality improves, and the detail of the image reveals. The image is usually examined at a closer distance, or is even magnified for close examination, which equivalently decreases the viewing distance. Thus, different sets of CSF weights are called for at different stages of the embedding. This adjustable application of visual frequency weighting is referred to as the visual progressive coding. Apparently, the fixed visual weighting can be viewed as a special case of the visual progressive coding.

L.1.1 Fixed Visual Weighting

In fixed visual weighting, a set of CSF weights, $\{w_i\}$, is chosen according to the final viewing condition, where w_i is the weight for the ith sub-band. The set of CSF weights can be incorporated in one of the following two ways.

L.1.1.1 Modify Quantization Step Size

At the encoder, the quantization step size q_i of the transform coefficients of the ith sub-band is adjusted inverse proportional to the CSF weight w_i. The smaller the CSF weight, the larger the quantization step size. The CSF-normalized quantization indices are then treated uniformly in the R-D optimization process, which is not modified to take into account any changes in the quantization step size. The CSF weights need not to be transmitted to the decoder. The information is included in the quantization step sizes that are explicitly transmitted for each sub-band. This approach needs to explicitly specify the quantizer so that it may not be very suitable for embedded coding, especially for embedded coding from lossy all the way to lossless.

L.1.1.2 Modify the embedded coding order

The quantization step sizes are not modified but the distortion weights fed into the R-D optimization are altered instead. This effectively controls the relative significance of including different numbers of bit-planes from the embedded bit stream of each code-block. The frequency-weighting table needs not to be transmitted explicitly. This approach is recommended since it produces similar results as in Annex L.1.1.1 and is compatible with lossless compression. This approach affects only the compressor and it is compatible with all quantization strategies, including implicit quantization.

L.1.2 Visual progressive coding (VIP)

If the visual frequency weights are to be changed during the embedded coding process, it is very clumsy to change the coefficient values or quantization step sizes. Furthermore, the performance of the subsequent entropy coder may degrade due to the changing statistics of the binary representation. An elegant way to implement the visual progressive coding (VIP) is to change, on the fly, the order in which code-block sub-bit-planes should appear in the overall embedded bit stream based on the visual weights, instead of changing the coefficient values or quantization step sizes. In other words, the coding order rather than the coding content is affected by the visual weights.

A series of visual weighting sets for different bit rate ranges are denoted as follows:

$$\begin{gathered}\text{Weighting set 0: } r(0), \text{ with } W(0) = \{w_0(0), w_1(0),. . . , w_n(0)\}; \qquad \text{L.1}\\ \text{Weighting set 1: } r(1), \text{ with } W(1) = \{w_0(1), w_1(1), . . . , w_n(1)\};\\ ...\\ \text{Weighting set m: } r(m), \text{ with } w(m) = \{w_0(m), w_1(m), . . . , w_n(m)\},\end{gathered}$$

where r(j) represents a bit-rate at which the weighting factors are changed, $r(0) < r(1) < ... < r(m)$, and $w_i(j)$ is the weight applied to sub-band i during bit rate range from r(j) to r(j+1). Each set of visual weights will take effect within a certain bit rate range. If m=0, i.e., there is only one set of visual weights, it degenerates to the fixed visual weighting case. The sets of visual weights, W(0) to W(m), will be used to determine the embedding order in their corresponding bit rate ranges. For high bit rate embedding, especially the embedded coding from lossy all the way to lossless, the final visual weights W(m) need to be all ones (as no weighting for lossless coding). The visual progressive coding can adjust the visual weights to achieve good visual quality for all bit rates.

The VIP weighting affects only the encoder and no signaling is required at the decoder.

The encoder is expected to compute the order in which code-block sub-bit-planes should appear in the layered hierarchy of the overall bit stream, based upon rate-distortion criteria. A simple implementation of progressive visual weighting changes the distortion metric progressively based on the visual weights during bit stream formation. Since bit stream formation is driven by post-compression R-D optimization, the progressively changing visual weights effectively control the embedding order of code-block sub-bit-planes on the fly.

L.1.3 Recommended frequency weighting tables

The following table specifies three sets of CSF weighting designed based on the CSF value at the mid-frequency of each sub-band. The viewing distance is supposed to be 1000, 2000, and 4000 pixels (e.g., corresponding to 10" for 100 dpi, 200 dpi, and 400 dpi print or display), respectively. Note that the tables are intended for a 5 level wavelet decomposition.

The table does not include the weight for the lowest frequency sub-band, nLL, which is always 1. Levels 1, 2, …, 5 denote the sub-band levels in low to high frequency order. (LH, HL, HH) denotes the three frequency orientations within each sub-band.

Table L-1 — Recommended frequency weighting

	Viewing distance (pixels)		
level	1000 (LH HL HH)	2000 (LH HL HH)	4000 (LH HL HH)
1	1 1 1	1 1 1	1 1 1
2	1 1 1	1 1 1	1.000000 1.000000 0.731668
3	1 1 1	1.000000 1.000000 0.727203	0.564344 0.564344 0.285968

Table L-1 — Recommended frequency weighting

level	Viewing distance (pixels) 1000 (LH HL HH)	2000 (LH HL HH)	4000 (LH HL HH)
4	1 1 0.727172	0.560841 0.560841 0.284193	0.179609 0.179609 0.043903
5	0.560805 0.560805 0.284173	0.178494 0.178494 0.043631	0.014774 0.014774 0.000573

Annex M

Patents

The following patents are claimed to be of relevance for the implementation of this Recommendation | International Standard.

Editor's note: this section is incomplete.

[1] C.K. Chui, R. Yi, "System and method for nested split coding of sparse data sets," U.S. Patent No. 5,748,116, **date**, and U.S. Patent No. 5,886,651, **date**.

[2] C.K. Chui, R. Yi, "System and method for tree ordered coding of sparse data sets," U.S. Patent No. 5,893,100, **date**.

[3] C.K. Chui, L. Zhong, and R. Yi, "System and method for scalable coding of sparse data sets," U.S. Patent. Allowed, **date**.

[4] C.K. Chui, "System and method for performing wavelet-like and inverse wavelets-like transforms of digital data using only add and bit split arithmetic operations," U.S. Patent No. 5,909,518, **date**.

[5] C. Christopoulos, J. Askelöf and M. Larsson, "A method and a system for coding Regions of Interest," Swedish Patent Application 9803454-9, 1998.

[6] D. Nister, C. Christopoulos, "Long filter Lossless region of interest coding," Swedish Patent Application 9800088-8, 1998.

[7] D. Nister, C. Christopoulos, "Lossless region of interest coding," Swedish Patent Application 9703690-9, 1997.

[8] M. Larsson, C. Christopoulos, M. Jändel, "Method and apparatus in transmission of images," Swedish Patent Application 9802193-4.

Annex N

Bibliography

N.1 General

[1] C. Christopoulos, A. Skodras, "The JPEG 2000," *JPEG2000 Tutorial presented in IEEE International Conference on Image Processing (ICIP 99)*, October 25-28, 1999, Kobe, Japan

[2] C. Christopoulos, T. Ebrahimi, A. Skodras, "The upcoming JPEG2000 standard," *Invited Tutorial to the 11th Portuguese Conference on Pattern Recognition (RECPAD 2000)*, Porto, Portugal, 11-12 May 2000

[3] T. Ebrahimi, C, Christopoulos, "JPEG 2000 standard: Still image compression scheme of 21st century," *JPEG2000 Tutorial to be presented in to the European Signal processing Conference (EUSIPCO 2000)*, Tampere, Finland, 5-8 September 2000.

N.2 Wavelet transform

[4] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using the wavelet transform," *IEEE Trans. Image Proc. 1*, pp. 205-220, April 1992.

[5] C. M. Brislawn, "Classification of nonexpansive symmetric extension transforms for multirate filter banks," *Appl. Comput. Harmonic. Analysis*, vol. 3, pp. 337-57, Month, 1996.

[6] C. Chrysafis, A. Ortega, "An algorithm for low memory wavelet image compression," *Proc. IEEE Int. Conference on Image Processing (ICIP)*, 24-28 October 1999, Kobe, Japan.

[7] C.K. Chui, *An Introduction to Wavelets*, Academic Press, Boston, 1992.

[8] C.K. Chui, *Wavelets: A Mathematical Tool for Signal Analysis*, SIAM Publ., Philadelphia, 1997.

[9] I. Daubechies, *Ten Lectures on Wavelets*, SIAM Publ., Philadelphia, 1992.

[10] I. Daubechies, W. Sweldens, "Factoring Wavelet Transforms into Lifting Steps," *J.Fourier Anal. Appl.*, vol. 4, no. 3, pp. 245-267, 1998.

[11] D. Le Gall and A. Tabatabai, "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques," IEEE International Conference on Acoustics, Speech and Signal Processing, New York, NY, pp. 761–765, 1988.

[12] S. Komatsu, K. Sezaki, and Y. Yasuda, "Reversible sub-band coding method of light/dark images," Electronic Information Communication Research Dissertation D-II, vol. J78-D-II, no. 3, pp. 429-436, 1995.

[13] M. Lighstone, E. Majani, S. K. Mitra, "Low bit-rate design considerations for wavelet-based image coding," Multidimensional Systems and Signal Processing, Vol. 8, No. 1, pp. 111-128, January 1997

[14] E. Majani, "Biorthogonal wavelets for image compression," SPIE Proceedings Vol. 2308, Visual Communications and Image Processing '94, pp. 478-488, September 25-29, 1994

[15] S. Mallat, *A Wavelet Tour of Signal Processing*, Second Edition, Academic Press, San Diego, 1999.

[16] A.V. Oppenheim, R.W. Schafer, *Discrete Time Signal Processing*, Prentice-Hall, 1989.

[17] W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmonic. Analysis,* vol. 3, no. 2, pp 186-200, 1996.

[18] W. Sweldens, "The lifting scheme: construction of second generation wavelets," *SIAM J.Math. Anal.*, vol. 29, no. 2, pp 511-546, 1997.

N.3 Quantization and Entropy coding

[19] E. Ordentlich, M. J. Weinberger, G. Seroussi, "A low-complexity modelling approach for embedded coding of wavelet coefficients," *Proc. of Data Compression Conference,* Snowbird, Utah, pp. 408-417, March 29-April 1, 1998.

[20] E. Ordentlich, D. Taubman, M.J. Weinberger, G. Seroussi, and M. Marcellin, "Memory Efficient Scalable Line-based Image Coding," *Proc. of Data Compression Conference*, Snowbird, Utah, pp. 218-227, March 29-31, 1999.

[21] D. Taubman, A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Trans. Image Processing*, vol. 3, pp. 572-88, September 1994.

[22] D. Taubman, "High performance scalable image compression with EBCOT", *Proc. IEEE Int. Conference on Image Processing (ICIP)*, 24-28 October 1999, Kobe, Japan.

[23] J.W. Woods, J. Naveen, "A filter based bit allocation scheme for subband compression of HDJV," IEEE Trans. on Image Processing, July 1992.

[24] A. Zandi, J. D. Allen, E. L. Scwhartz, M. Boliek, "CREW: Compression with reversible embedded wavelets," Proc. of Data Compression Conference, Snowbird, UT, pp. 212-21, March 1995.

N.4 Region of Interest coding

[25] E. Atsumi, N. Farvardin, “Lossy/lossless region-of-interest image coding based on set partitioning in hierarchical trees,” Proc. IEEE International Conference on Image Processing (ICIP-98), pp. 87-91, 4-7 October 1998, Chicago, Illinois

[26] Nister D. and Christopoulos C., “Lossless Region of Interest with a naturally progressive still image coding algorithm,” Proceedings of IEEE International Conference on Image Processing (ICIP 98), pp. 856-860, 4-7 October 1998, Chicago, Illinois,

[27] D. Nister, C. Christopoulos, “Lossless region of interest with embedded wavelet image coding,” Signal Processing, Vol. 78, No. 1, pp. 1-17, October 1999.

[28] D. Santa Cruz, M. Larsson, J. Askelof, T. Ebrahimi, C. Christopoulos, “Region of Interest coding in JPEG2000 for interactive client/server applications,” IEEE International Workshop on Multimedia Signal Processing, Copenhagen, Denmark, 13-15 September 1999.

N.5 Visual frequency weighting

[29] M. Albanesi and S. Bertoluzza, “Human vision model and wavelets for high-quality image compression,” Proc. of 5th Int. Conference in Image Processing and its Applications, Edinburgh, UK, no. 410, pp. 311-315, 4-6 July 1995.

[30] M. Eckert, “Lossy compression using wavelets, block DCT, and lapped orthogonal transforms optimised with a perceptual model,” SPIE vol. 3031, pp. 339-350, month 1997.

[31] P. Jones, S. Daly, R. Gaborski and M. Rabbani, “Comparative study of wavelet and DCT decompositions with equivalent quantization and encoding strategies for medical images,” Proceedings of Conference on Medical Imaging, SPIE vol. 2431, pp. 571-582, month 1995.

[32] T. O’Rourke and R. Stevenson, “Human visual system based wavelet decomposition for image compression,” J. VCIP V. 6, pp. 109-121, month 1995

[33] Watson, G. Yang, J. Solomon, and J. Villasenor, “Visibility of wavelet quantization noise,” IEEE Trans. on Image Proc., vol. 6, pp. 1164-1175, month 1997.

N.6 Post-processing

[34] M. Shen and C.-C. Jay Kuo, “Artifact Reduction in Low Bit Rate Wavelet Coding with Robust Non-linear Filtering,” Proc. of IEEE 1998 Workshop on Multimedia Signal Processing (MMSP-98), Los Angeles, California, 7-9 December 1998.

N.7 Error resilience

[35] H. Man, F. Kossentini and M. Smith, “A Family of Efficient and Channel Error Resilient Wavelet/Subband Image Coders,” Special issue of the IEEE Transactions on Circuits and Systems for Video Technology on Interactive Multimedia, 9(2), February 1999.

Index