INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.800

(08/2002)

SERIES T: TERMINALS FOR TELEMATIC SERVICES

Information technology – JPEG 2000 image coding system: Core coding system

ITU-T Recommendation T.800

INTERNATIONAL STANDARD ISO/IEC 15444-1
ITU-T RECOMMENDATION T.800

Information technology – JPEG 2000 image coding system: Core coding system

Summary

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

Source

ITU-T Recommendation T.800 was prepared by ITU-T Study Group 16 (2001-2004) and approved on 29 August 2002. An identical text is also published as ISO/IEC 15444-1.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

Page

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – JPEG 2000 image coding system: Core coding system

1 Scope

This Recommendation | International Standard defines a set of lossless (bit-preserving) and lossy compression methods for coding bi-level, continuous-tone grey-scale, palletized color, or continuous-tone 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 (1992) | ISO/IEC 10918-1:1994, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines*.
- ITU-T Recommendation T.88 (2000) | ISO/IEC 14492:2001, *Information technology – Lossy/lossless coding of bi-level images*.
- ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*.
- ISO 8859-15:1999, *Information technology – 8-bit single-byte coded graphic character sets – Part 15: Latin alphabet No. 9*.
- ITU-T Recommendation T.84 (1996) | ISO/IEC 10918-3:1997, *Information technology – Digital compression and coding of continuous-tone still images: Extensions*.
- ITU-T Recommendation T.84 (1996)/Amd.1 (1999) | ISO/IEC 10918-3:1997/Amd.1:1999, *Information technology – Digital compression and coding of continuous-tone still images: Extensions – Amendment 1: Provisions to allow registration of new compression types and versions in the SPIFF header*.
- ITU-T Recommendation T.86 (1998) | ISO/IEC 10918-4:1999, *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 (1998) | ISO/IEC 14495-1:2000, *Lossless and near-lossless compression of continuous-tone still images – Baseline*.

2.2 Additional references

- Specification ICC.1:1998-09, *File format for Color Profiles*.
- IEC 61966-2-1:1999, *Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management – Default RGB colour space – sRGB*.
- W3C REC-xml-19980210, *Extensible Markup Language (XML 1.0)*.
- IETF RFC 2279 (1998), UTF-8, *a transformation format of ISO 10646*.

– ISO/IEC 11578:1996, *Information technology – Open Systems Interconnection – Remote Procedure Call*.
– IEC 61966-2-1:1999/Amd.1:2003, *Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management – Default RGB colour space – sRGB*.

3 Definitions

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

3.1 $\lfloor x \rfloor$, **floor function**: This indicates the largest integer not exceeding x.

3.2 $\lceil x \rceil$, **ceiling function**: This indicates the smallest integer not exceeded by x.

3.3 **5-3 reversible filter**: A particular filter pair used in the wavelet transformation. This reversible filter pair has 5 taps in the low-pass and 3 taps in the high-pass.

3.4 **9-7 irreversible filter**: A particular filter pair used in the wavelet transformation. This irreversible filter pair has 9 taps in the low-pass and 7 taps in the high-pass.

3.5 **AND**: Bit wise AND logical operator.

3.6 **arithmetic coder**: An entropy coder that converts variable length strings to variable length codes (encoding) and visa versa (decoding).

3.7 **auxiliary channel**: A channel that is used by the application outside the scope of colourspace conversion. For example, an opacity channel or a depth channel would be an auxiliary channel.

3.8 **bit**: A contraction of the term "binary digit"; a unit of information represented by a zero or a one.

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

3.10 **bit stream**: The actual sequence of bits resulting from the coding of a sequence of symbols. It does not include the markers or marker segments in the main and tile-part headers or the EOC marker. It does include any packet headers and in stream markers and marker segments not found within the main or tile-part headers.

3.11 **big endian**: The bits of a value representation occur in order from most significant to least significant.

3.12 **box**: A portion of the file format defined by a length and unique box type. Boxes of some types may contain other boxes.

3.13 **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.

3.14 **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.

3.15 **byte**: Eight bits.

3.16 **channel**: One logical component of the image. A channel may be a direct representation of one component from the codestream, or may be generated by the application of a palette to a component from the codestream.

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

3.18 **codestream**: A collection of one or more bit streams and the main header, tile-part headers, and the EOC required for their decoding and expansion into image data. This is the image data in a compressed form with all of the signalling needed to decode.

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

3.20 **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 (or all remaining rows if less than four) and spanning the width of the code-block. Each stripe is processed column by column from top to bottom and from left to right.

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

3.22 **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. The result of each pass (after arithmetic coding, if selective arithmetic coding bypass is not used) is a stream of compressed image data.

3.23 **coefficient**: The values that are result of a transformation.

3.24 **colour channel**: A channel that functions as an input to a colour transformation system. For example, a red channel or a greyscale channel would be a colour channel.

3.25 **component**: A two-dimensional array of samples. A image typically consists of several components, for instance representing red, green, and blue.

3.26 **compressed image data**: Part or all of a bit stream. Can also refer to a collection of bit streams in part or all of a codestream.

3.27 **conforming reader**: An application that reads and interprets a JP2 file correctly.

3.28 **context**: Function of coefficients previously decoded and used to condition the decoding of the present coefficient.

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

3.30 **context vector**: The binary vector consisting of the significance states of the coefficients included in a context.

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

3.32 **decoding process**: A process which takes as its input all or part of a codestream and outputs all or part of a reconstructed image.

3.33 **decomposition level**: A collection of wavelet sub-bands where each coefficient has the same spatial impact or span with respect to the source component samples. These include the HL, LH, and HH sub-bands of the same two dimensional sub-band decomposition. For the last decomposition level the LL sub-band is also included.

3.34 **delimiting markers and marker segments**: Markers and marker segments that give information about beginning and ending points of structures in the codestream.

3.35 **discrete wavelet transformation (DWT)**: A transformation that iteratively transforms one signal into two or more filtered and decimated signals corresponding to different frequency bands. This transformation operates on spatially discrete samples.

3.36 **encoder**: An embodiment of an encoding process.

3.37 **encoding process**: A process that takes as its input all or part of a source image data and outputs a codestream.

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

3.39 **fixed information markers and fixed information marker segments**: Markers and marker segments that offer information about the original image.

3.40 **functional markers and functional marker segments**: Markers and marker segments that offer information about the coding procedures.

3.41 **grid resolution**: The spatial resolution of the reference grid, specifying the distance between neighboring points on the reference grid.

3.42 **guard bits**: Additional most significant bits that have been added to sample data.

3.43 **header**: Either a part of the codestream that contains only markers and marker segments (main header and tile-part header) or the signalling part of a packet (packet header).

3.44 **HH sub-band**: The sub-band obtained by forward horizontal high-pass filtering and vertical high-pass filtering. This subband contributes to reconstruction with inverse vertical high-pass filtering and horizontal high-pass filtering.

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

3.46 **image**: The set of all components.

3.47 **image area**: A rectangular part of the reference grid, registered by offsets from the origin and the extent of the reference grid.

3.48 **image area offset**: The number of reference grid points down and to the right of the reference grid origin where the origin of the image area can be found.

3.49 **image data**: The components and component samples making up an image. Image data can refer to either the source image data or the reconstructed image data.

3.50 **in-bit-stream markers and in-bit-stream marker segments**: Markers and marker segments that provide error resilience functionality.

3.51 **informational markers and informational marker segments**: Markers and marker segments that offer ancillary information.

3.52 **irreversible**: A transformation, progression, system, quantization, or other process that, due to systemic or quantization error, disallows lossless recovery. An irreversible process can only lead to lossy compression.

3.53 **JP2 file**: The name of a file in the file format described in this Recommendation | International Standard. Structurally, a JP2 file is a contiguous sequence of boxes.

3.54 **JPEG**: Used to refer globally to the encoding and decoding process of the following Recommendations | International Standards:

- ITU-T Rec. T.81 | ISO/IEC 10918-1, Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines;
- ITU-T Rec. T.83 | ISO/IEC 10918-2, Information technology – Digital compression and coding of continuous-tone still images: Compliance testing;
- ITU-T Rec. T.84 | ISO/IEC 10918-3, Information technology – Digital compression and coding of continuous-tone still images: Extensions;
- ITU-T Rec. T.84 | ISO/IEC 10918-3/Amd.1, Information technology – Digital compression and coding of continuous-tone still images: Extensions – Amendment 1: Provisions to allow registration of new compression types and versions in the SPIFF header;
- ITU-T Rec. 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).

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

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

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

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

3.59 **lossless**: A descriptive term for the effect of the overall encoding and decoding processes in which the output of the decoding process is identical to the input to the encoding process. Distortion-free restoration can be assured. All of the coding processes or steps used for encoding and decoding are reversible.

3.60 **lossy**: A descriptive term for the effect of the overall encoding and decoding processes in which the output of the decoding process is not identical to the input to the encoding process. There is distortion (measured mathematically). At least one of the coding processes or steps used for encoding and decoding is irreversible.

3.61 **magnitude refinement pass**: A type of coding pass.

3.62 **main header**: A group of markers and marker segments at the beginning of the codestream that describe the image parameters and coding parameters that can apply to every tile and tile-component.

3.63 **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).

3.64 **marker segment**: A marker and associated (not empty) set of parameters.

3.65 **mod**: mod(y,x) = z, where z is such that $0 \le z < x$, and such that y – z is a multiple of x.

3.66 **packet**: A part of the bit stream comprising a packet header and the compressed image data from one layer of one precinct of one resolution level of one tile-component.

3.67 **packet header**: Portion of the packet that contains signalling necessary for decoding that packet.

3.68 **pointer markers and pointer marker segments**: Markers and marker segments that offer information about the location of structures in the codestream.

3.69 **precinct**: A one rectangular region of a transformed tile-component, within each resolution level, used for limiting the size of packets.

3.70 **precision**: Number of bits allocated to a particular sample, coefficient, or other binary numerical representation.

3.71 **progression**: The order of 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 sample fidelity.

3.72 **quantization**: A method of reducing the precision of the individual coefficients to reduce the number of bits used to entropy-code them. This is equivalent to division while compressing and multiplying while decompressing. Quantization can be achieved by an explicit operation with a given quantization value or by dropping (truncating) coding passes from the codestream.

3.73 **raster order**: A particular sequential order of data of any type within an array. The raster order starts with the top left data point and moves to the immediate right data point, and so on, to the end of the row. After the end of the row is reached the next data point in the sequence is the left-most data point immediately below the current row. This order is continued to the end of the array.

3.74 **reconstructed image**: An image that is the output of a decoder.

3.75 **reconstructed sample**: A sample 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.

3.76 **reference grid**: A regular rectangular array of points used as a reference for other rectangular arrays of data. Examples include components and tiles.

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

3.78 **region of interest (ROI)**: A collections of coefficients that are considered of particular relevance by some user-defined measure.

3.79 **resolution level**: Equivalent to decomposition level with one exception: the LL sub-band is also a separate resolution level.

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

3.81 **sample**: One element in the two-dimensional array that comprises a component.

3.82 **segmentation symbol**: A special symbol coded with a uniform context at the end of each coding pass for error resilience.

3.83 **selective arithmetic coding bypass**: A coding style where some of the code-block passes are not coded by the arithmetic coder. Instead the bits to be coded are appended directly to the bit stream without coding.

3.84 **shift**: Multiplication or division of a number by powers of two.

3.85 **sign bit**: A bit that indicates whether a number is positive (zero value) or negative (one value).

3.86 **sign-magnitude notation**: A binary representation of an integer 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 sign bit.

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

3.88 **significance state**: State of a coefficient at a particular bit-plane. If a coefficient, in sign-magnitude notation, has the first magnitude 1 bit at, or before, the given bit-plane it is considered "significant". If not, it is considered "insignificant".

3.89 **source image**: An image used as input to an encoder.

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

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

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

3.93 **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.

3.94 **tile**: A rectangular array of points on the reference grid, registered with and offset from the reference grid origin and defined by a width and height. The tiles which overlap are used to define tile-components.

3.95 **tile-component**: All the samples of a given component in a tile.

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

3.97 **tile-part**: A portion of the codestream with compressed image data for some, or all, of a tile. The tile-part includes at least one, and up to all, of the packets that make up the coded tile.

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

3.99 **tile-part index**: The index of the current tile-part ranging from zero to the number of tile-parts minus one in a given tile.

3.100 **transformation**: A mathematical mapping from one signal space to another.

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

3.102 **XOR**: Exclusive OR logical operator.

4 Abbreviations and symbols

4.1 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
- **ICT** Irreversible Component Transform
- **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 Rec. T.81 | ISO/IEC 10918-1, ITU-T Rec. T.83 | ISO/IEC 10918-2, ITU-T Rec. T.84 | ISO/IEC 10918-3 and ITU-T Rec. T.87 | ISO/IEC 14495.
- **JURA** JPEG Utilities Registration Authority
- **1D-DWT** One-dimensional Discrete Wavelet Transformation
- **FDWT** Forward Discrete Wavelet Transformation
- **IDWT** Inverse Discrete Wavelet Transformation

LSB	Least Significant Bit
MSB	Most Significant Bit
PCS	Profile Connection Space
RCT	Reversible Component Transform
ROI	Region Of Interest
SNR	Signal-to-Noise Ratio
UCS	Universal Character Set
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF-8	UCS Transformation Format 8
UUID	Universal Unique Identifier
XML	Extensible Markup Language
W3C	World-Wide Web Consortium

4.2 Symbols

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

0x----	Denotes a hexadecimal number
\nnn	A three-digit number preceded by a backslash indicates the value of a single byte within a character string, where the three digits specify the octal value of that byte
ε_b	Exponent of the quantization value for a sub-band defined in QCD and QCC
μ_b	Mantissa of the quantization value for a sub-band defined in QCD and QCC
M_b	Maximum number of bit-planes coded in a given code-block
N_L	Number of decomposition levels as defined in COD and COC
R_b	Dynamic range of a component sample as defined in SIZ
COC	Coding style component marker
COD	Coding style default marker
COM	Comment marker
CRG	Component registration marker
EPH	End of packet header marker
EOC	End of codestream marker
PLM	Packet length, main header marker
PLT	Packet length, tile-part header marker
POC	Progression order change marker
PPM	Packed packet headers, main header marker
PPT	Packed packet headers, tile-part header marker
QCC	Quantization component marker
QCD	Quantization default marker
RGN	Region-of-interest marker
SIZ	Image and tile size marker
SOC	Start of codestream marker
SOP	Start of packet marker
SOD	Start of data marker
SOT	Start of tile-part marker
TLM	Tile-part lengths marker

5 General description

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

The division of both the original image data and the compressed image data in a number of ways leads to the ability to extract image data from the compressed image data to form a reconstructed image with lower resolution or lower precision, or regions of the original image. 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 Recommendation | International Standard 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.

In general, this Recommendation | International Standard deals with three domains: spatial (samples), transformed (coefficients), and compressed image data. Some entities (e.g., tile-component) have meaning in all three domains. Other entities (e.g., code-block or packet) have meaning in only one domain (e.g., transformed or compressed image data, respectively). The splitting of an entity into other entities in the same domain (e.g., component to tile-components) is described separately for each of the domains.

5.1 Purpose

There are four 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 a codestream.
- Decoder: An embodiment of a decoding 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 syntax: A representation of the compressed image data that includes all parameter specifications required by the decoding process.
- Optional file format: The optional file format is for exchange between application environments. The codestream can be used by other file formats or stand-alone without this file format.

5.2 Codestream

The codestream is a linear stream of bits from the first bit to the last bit. For convenience, it can be divided into (8-bit) bytes, starting with the first bit of the codestream, with the "earlier" bit in a byte viewed as the most significant bit of the byte when given e.g., a hexadecimal representation. This byte stream may be divided into groups of consecutive bytes. The hexadecimal value representation is sometimes implicitly assumed in the text when describing bytes or group of bytes that do not have a "natural" numeric value representation.

5.3 Coding principles

The main procedures for this Recommendation | International Standard are shown in Figure 5-1. This shows the decoding order only. The compressed image data is already conceptually assigned to portions of the image data. Procedures are presented in the Annexes in the order of the decoding process. The coding process is summarized below.

NOTE 1 – Annexes A through I are considered normative to this Recommendation | International Standard. Certain denoted sub-clauses and notes and all examples are informative, however.

Many images have multiple components. This Recommendation | International Standard has a multiple component transformation to decorrelate three components. This is the only function in this Recommendation | International Standard that relates components to each other. (See Annex G.)

The image components may be divided into tiles. These tile-components are rectangular arrays that relate to the same portion of each of the components that make up the image. Thus, tiling of the image actually creates tile-components that can be extracted or decoded independently of each other. This tile independence provides one of the methods for extracting a region of the image. (See Annex B.)

The tile-components are decomposed into different decomposition levels using a wavelet transformation. 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-components. The coefficients provide frequency information about a local area, rather than across the entire image like the Fourier transformation. That is, a small number of coefficients completely describe a single sample. A decomposition level is related to the next decomposition level by a spatial factor of two. That is, each successive decomposition level of the sub-bands 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 subset of these sub-bands. (See Annex F.)

Although there are as many coefficients as there are 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 (Annex E). Additional processing by the entropy coder reduces the number of bits required to represent these quantized coefficients, sometimes significantly compared to the original image. (See Annexes C, D, and B.)

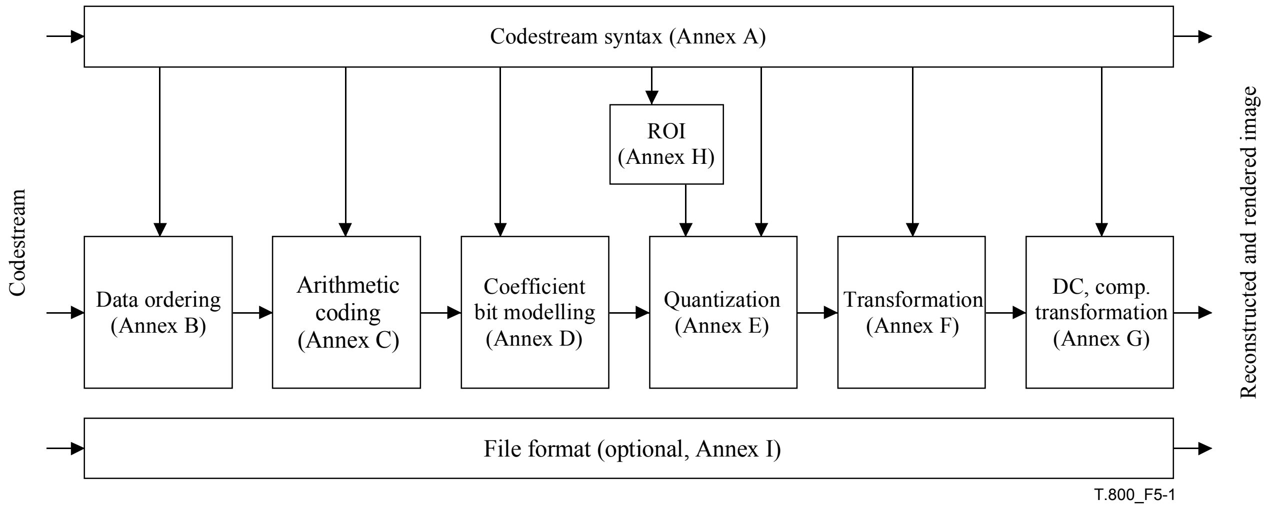


Figure 5-1 – Specification block diagram

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 compressed image data. (See Annex D.) An arithmetic coder uses this contextual information, and its internal state, to decode a compressed bit stream. (See Annex C.) Different termination mechanisms allow different levels of independent extraction of this coding pass compressed image data.

The bit stream compressed image data created from these coding passes is grouped in layers. Layers are arbitrary groupings of coding passes from code-blocks. (See Annex B.)

NOTE 2 – Although there is great flexibility in layering, the premise is that each successive layer contributes to a higher quality image.

Sub-band coefficients at each resolution level are partitioned into rectangular areas called precincts. (See Annex B.)

Packets are a fundamental unit of the compressed codestream. A packet contains compressed image data from one layer of a precinct of one resolution level of one tile-component. Packets provide another method for extracting a spatial region independently from the codestream. These packets are interleaved in the codestream using a few different methods. (See Annex B.)

A mechanism is provided that allows the compressed image 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 D.5 and J.7.)

The codestream relating to a tile, organized in packets, are arranged in one, or more, tile-parts. A tile-part header, comprised of 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 offers similar information as well as information about the original image. (See Annex A.)

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

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

- The components of the image are divided into rectangular tiles. The tile-component is the basic unit of the original or reconstructed image.
- Performing the wavelet transformation on a tile-component creates decomposition levels.
- These decomposition levels are made up of sub-bands of coefficients that describe the frequency characteristics of local areas (rather than across the entire tile-component) of the tile-component.
- The sub-bands of coefficients are quantized and collected into rectangular arrays of code-blocks.
- Each bit-plane of the coefficients in a code-block are entropy coded in three types of coding passes.
- Some of the coefficients can be coded first to provide a region of interest.

At this point the image data is fully converted to compressed image data. The procedures that reassemble these bit stream units into the codestream are the following:

- The compressed image data from the coding passes are collected in layers.
- Packets are composed compressed image data from one precinct of a single layer of a single resolution level of a single tile-component. The packets are the basic unit of the compressed image 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 some orders.
- The codestream has a main header at the beginning that describes the original image and the various decomposition and coding styles.
- The optional file format describes the meaning of the image and its components in the context of the application.

6 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. 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, that conform to the codestream syntax specified in Annex A.

7 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Annexes A through H describe and specify the decoding process. All decoding processes are normative.

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 that conform to the codestream syntax specified in Annex A to a reconstructed image.

There is no normative or required implementation for the encoder or decoder. In some cases, the descriptions use particular implementation techniques for illustrative purposes only.

7.1 Codestream syntax requirements

Annex A describes the codestream syntax that defines the coded representation of compressed image data for exchange between application environments. Any compressed image data shall comply with the syntax and code assignments appropriate for the coding processes defined in the Recommendation | International Standard.

This Recommendation | International Standard does not include a definition of compliance or conformance. The parameter values of the syntax described in Annex A are not intended to portray the capabilities required to be compliant.

7.2 Optional file format requirements

Annex I describes the optional file format containing metadata about the image in addition to the codestream. This data allows, for example, screen display or printing at a specific resolution. The optional file format, when used, shall comply with the file format syntax and code assignments appropriate for the coding processes defined in the Recommendation | International Standard.

8 Implementation requirements

There is no normative or required implementation for this Recommendation | International Standard. In some cases, the descriptions use particular implementation techniques for illustrative purposes only.

Annex A

Codestream syntax

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

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.

This annex specifies the marker and marker segment syntax and semantics defined by this Recommendation | International Standard. These markers and marker segments provide codestream information for this Recommendation | International Standard. Further, this annex provides a marker and marker segment syntax that is designed to be used in future specifications that include this Recommendation | International Standard as a normative reference.

This Recommendation | International Standard does not include a definition of compliance or conformance. The parameter values of the syntax described in this annex are not intended to portray the capabilities required to be compliant.

A.1 Markers, marker segments, and headers

This Recommendation | International Standard uses markers and marker segments to delimit and signal the characteristics of the source image and 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 file format. A minimal file format is offered in Annex I.

Main and tile-part headers are collections of markers and marker segments. The main header is found at the beginning of the codestream. The tile-part headers are found at the beginning of each tile-part (see below). Some markers and marker segments are restricted to only one of the two types of headers while others can be found in either.

Every marker is two bytes long. The first byte consists of a single 0xFF byte. The second byte denotes the specific marker and can have any value in the range 0x01 to 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 and shall be regarded as reserved unless specifically used.

A marker segment includes a marker and associated parameters, called marker segment parameters. In every marker segment the first two bytes after the marker shall be an unsigned value that denotes the length in bytes of the marker segment parameters (including the two bytes of this length parameter but not the two bytes of the marker itself). When a marker segment that is not specified in the Recommendation | International Standard appears in a codestream, the decoder shall use the length parameter to discard the marker segment.

A.1.1 Types of markers and marker segments

Six types of markers and marker segments are used: delimiting, fixed information, functional, in-bit stream, pointer, and informational. Delimiting markers and marker segments are used to frame the main and tile-part headers and the bit-stream data. Fixed information marker segments give required information about the image. The location of these marker segments, like delimiting marker and marker segments, is specified. Functional marker segments are used to describe the coding functions used. In-bit-stream markers and marker segments are used for error resilience. Pointer marker segments provide specific offsets in the bit stream. Informational marker segments provide ancillary information.

A.1.2 Syntax similarity with ITU-T Rec. T.81 | ISO/IEC 10918-1

The marker and marker segment syntax uses the same construction as defined in ITU-T Rec. T.81 | ISO/IEC 10918-1.

The marker range 0xFF30 to 0xFF3F is reserved by this Recommendation | International Standard for markers without marker segment parameters. Table A.1 shows in which specification these markers and marker segments are defined.

Table A.1 – Marker definitions

Marker code range	Standard definition
0xFF00, 0xFF01, 0xFFFE, 0xFFC0 to 0xFFDF	Defined in ITU-T Rec. T.81 | ISO/IEC 10918-1
0xFFF0 to 0xFFF6	Defined in ITU-T Rec. T.84 | ISO/IEC 10918-3
0xFFF7 to 0xFFF8	Defined in ITU-T Rec. T.87 | ISO/IEC 14495-1
0xFF4F to 0xFF6F, 0xFF90 to 0xFF93	Defined in this Recommendation | International Standard
0xFF30 to 0xFF3F	Reserved for definition as markers only (no marker segments)
	All other values reserved

A.1.3 Marker and marker segment and codestream rules

– Marker segments, and therefore the main and tile-part headers, are a multiple of 8 bits (one byte). Further, the bit stream data between the headers and before the EOC marker (see A.4.4) 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 they belong.

– All marker segments in the main header apply to the whole image unless specifically overridden by markers or marker segments in a tile-part header.

– Delimiting and fixed information marker and marker segments must appear at specific points in the codestream.

– The marker segments shall 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, if necessary.

– All parameter values in marker segments are big endian.

– Marker segments can appear in any order in a given header. Exceptions are the delimiting markers and marker segments and the fixed information marker segments.

– All markers with the marker code between 0xFF30 and 0xFF3F have no marker segment parameters. They shall be skipped by the decoder.

– Some marker segments have values assigned to groups of bits within a parameter. In some cases there are bits, denoted by "x," that are not assigned a value for any field within a parameter. The codestream shall contain a value of zero for all such bits. The decoder shall ignore these bits.

NOTE – The markers in the range 0xFF30 to 0xFF3F may be used by future extensions. They may or may not be skipped by a decoder without ramification.

A.1.4 Key to graphical descriptions (informative)

Each marker segment is described in terms of its 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 the three-letter code of the marker associated with the marker segment. The parameter symbols have capital letter designations followed by the marker's symbol in lower-case letters. A rectangle is used to indicate a parameter's location in the marker segment. The width of the rectangle is proportional to the number of bytes of the parameter. A shaded rectangle (diagonal stripes) indicates that the parameter is of varying size. Two parameters with superscripts and a grey area between indicate a run of several of these parameters.

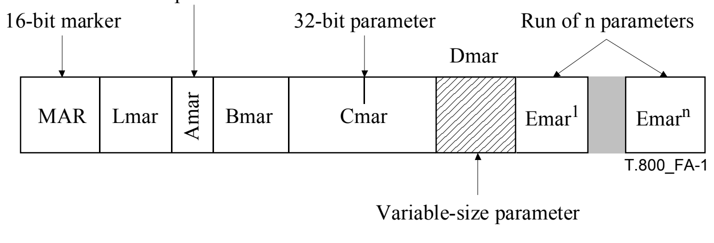


Figure A.1 – Example of the marker segment description figures

The figure is followed by a list that describes the meaning of each parameter in the marker segment. If parameters are repeated, the length and nature of the run of parameters is defined. As an example, in Figure A.1, the first rectangle represents the marker with the symbol MAR. The second rectangle represents the length parameter. Parameters Amar, Bmar, Cmar, and Dmar are 8-, 16-, 32-bit and variable length respectively. The notation $Emar^i$ implies that there are *n* different parameters, $Emar^i$, in a row.

After the list is a table that either describes the allowed parameter values or provides references to other tables that describe these values. Tables for individual parameters are provided to describe any parameter without a simple numerical value. In some cases these parameters are described by a bit value in a bit field. In this case, an "x" is used to denote bits that are not included in the specification of the parameter or sub-parameter in the corresponding row of the table.

Some marker segment parameters are described using the notation "Sxxx" and "SPxxx" (for a marker symbol, XXX). The Sxxx parameter selects between many possible states of the SPxxx parameter. 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 which information is provided by which marker and marker segments.

Table A.2 – List of markers and marker segments

	Symbol	Code	Main header	Tile-part header
Delimiting markers and marker segments				
Start of codestream	SOC	0xFF4F	required[a]	not allowed
Start of tile-part	SOT	0xFF90	not allowed	required
Start of data	SOD	0xFF93	not allowed	last marker
End of codestream	EOC	0xFFD9	not allowed	not allowed
Fixed information marker segments				
Image and tile size	SIZ	0xFF51	required	not allowed
Functional marker segments				
Coding style default	COD	0xFF52	required	optional
Coding style component	COC	0xFF53	optional	optional
Region-of-interest	RGN	0xFF5E	optional	optional
Quantization default	QCD	0xFF5C	required	optional
Quantization component	QCC	0xFF5D	optional	optional
Progression order change[b)]	POC	0xFF5F	optional	optional
Pointer marker segments	TLM	0xFF55	optional	not allowed
Tile-part lengths	PLM	0xFF57	optional	not allowed
Packet length, main header	PLT	0xFF58	not allowed	optional
Packet length, tile-part header	PPM	0xFF60	optional	not allowed
Packed packet headers, main header[c)]	PPT	0xFF61	not allowed	optional
Packed packet headers, tile-part header[c)]	TLM	0xFF55	optional	not allowed
In-bit-stream markers and marker segments				
Start of packet	SOP	0xFF91	not allowed	not allowed in tile-part header, optional in-bit stream
End of packet header	EPH	0xFF92	optional inside PPM marker segment	optional inside PPT marker segment or in-bit stream
Informational marker segments				
Component registration	CRG	0xFF63	optional	not allowed
Comment	COM	0xFF64	optional	optional

a) "required" means the marker or marker segment shall be in this header; "optional" means it may be used.

b) The POC marker segment is required if there are progression order changes.

c) Either the PPM or PPT marker segment is required if the packet headers are not distributed in the bit stream. If the PPM marker segment is used then PPT marker segments shall not be used, and vice versa.

Table A.3 – Information in the marker segments

Information	Marker segment
Capabilities Image area size or reference grid size (height and width) Tile size (height and width) Number of components Component precision Component mapping to the reference grid (sub-sampling)	SIZ
Tile index Tile-part data length	SOT, TLM
Progression order Number of layers Multiple component transformation used	COD
Coding style Number of decomposition levels Code-block size Code-block style Wavelet transformation Precinct size	COD, COC
Region-of-interest shift	RGN
No quantization Quantization derived Quantization expounded	QCD, QCC
Progression starting point Progression ending point Progression order default	POC
Error resilience	SOP
End of packet header	EPH
Packet headers	PPM, PPT
Packet lengths	PLM, PLT
Component registration Optional information	CRG COM

A.3 Construction of the codestream

Figure A.2 shows the construction of the codestream. Figure A.3 shows the main header construction. All of the solid lines show required marker segments. The following markers and marker segments are required to be in a specific location: SOC, SIZ, SOT, SOD, and EOC. The dashed lines show optional or possibly not required marker segments. Figure A.4 shows the construction of the first tile-part header in a given tile. Figure A.5 shows the construction of a tile-part header other than the first in a tile.

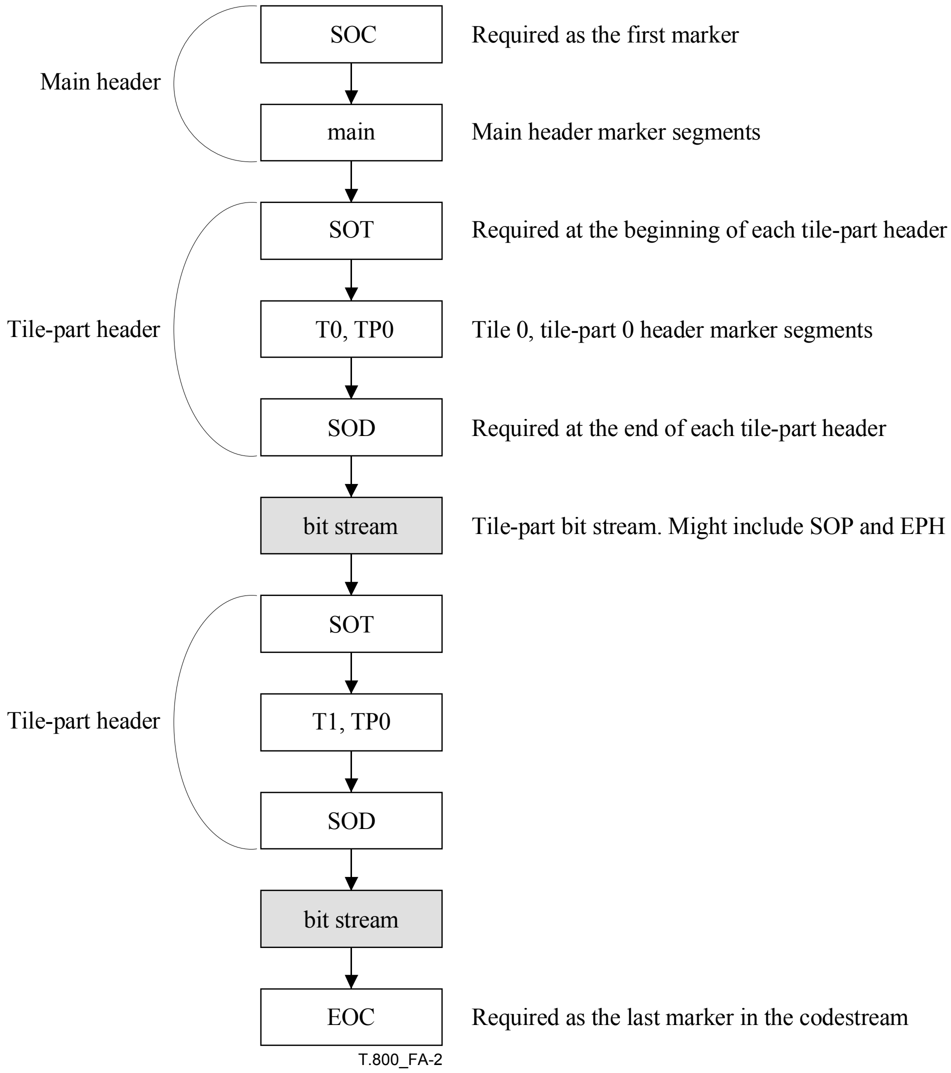


Figure A.2 – Construction of the codestream

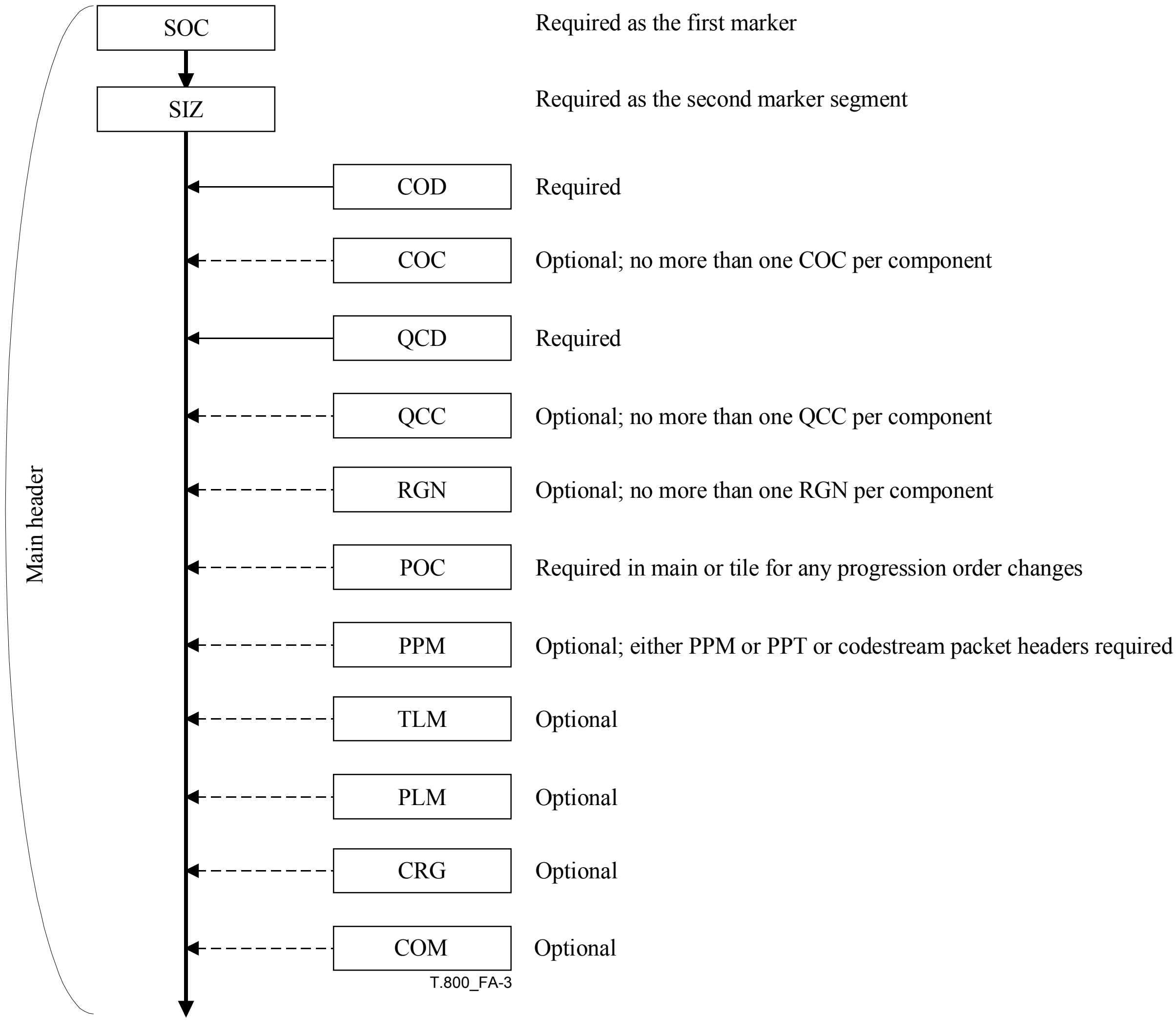


Figure A.3 – Construction of the main header

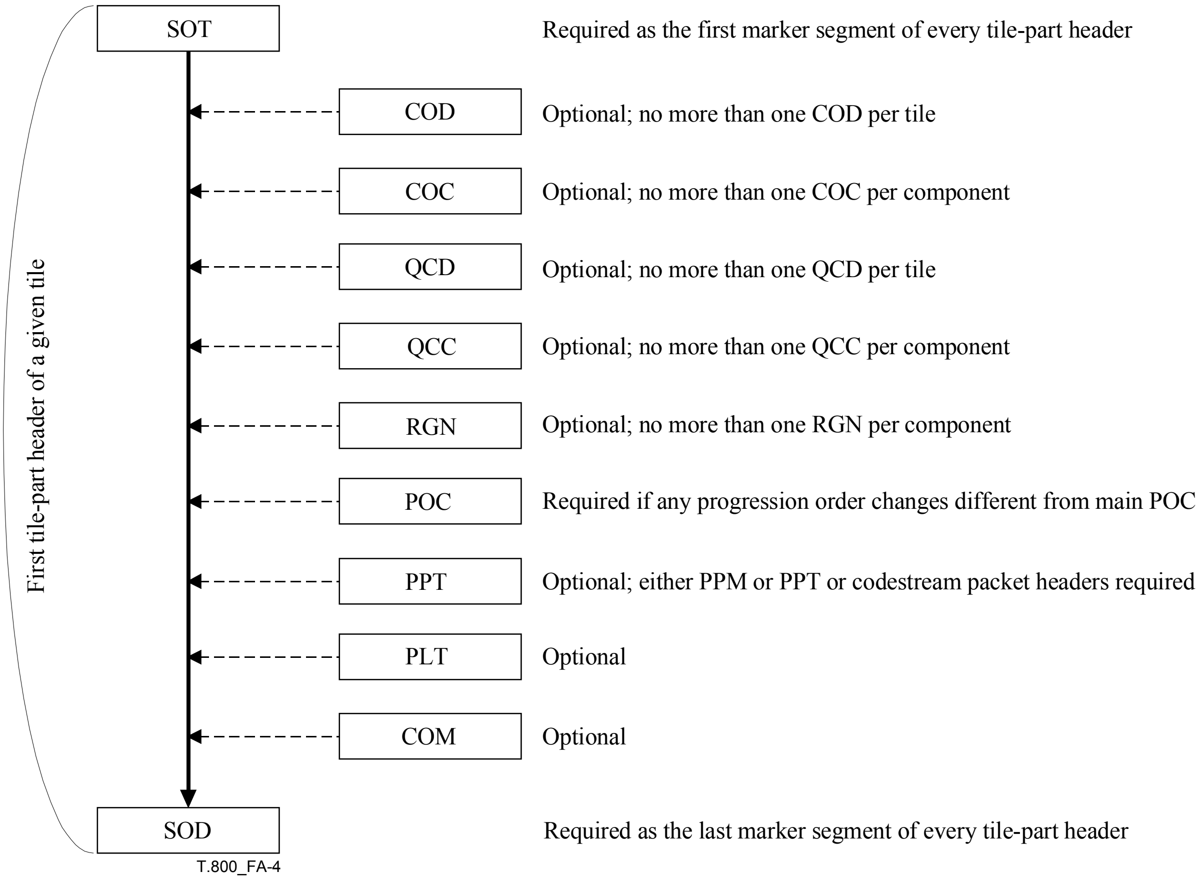


Figure A.4 – Construction of the first tile-part header of a given tile

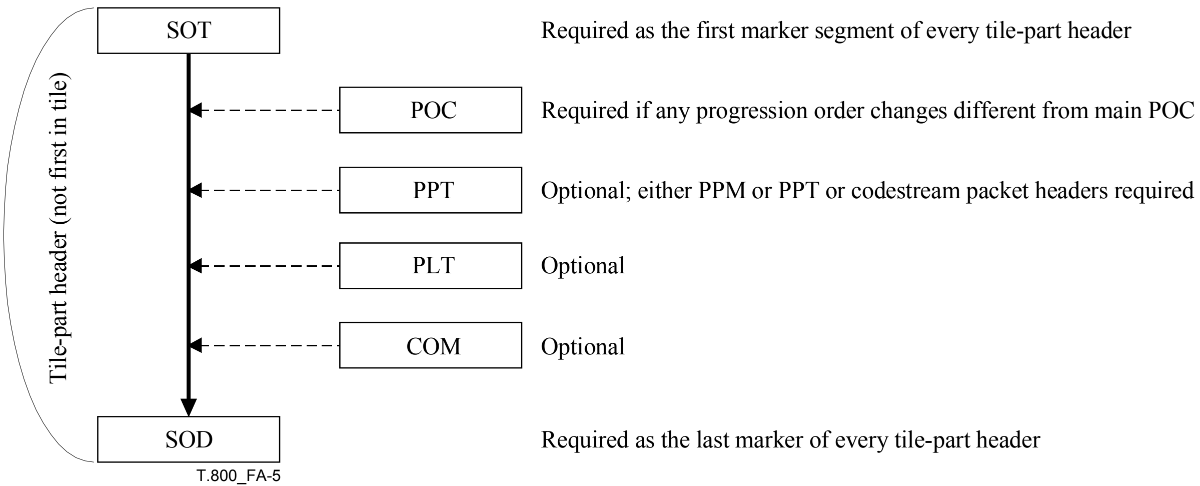


Figure A.5 – Construction of a non-first tile-part header

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 chrominance 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-part headers.

The POC marker segment appearing in the main header is used for all tiles unless a different POC appears in the tile-part header.

With the exceptions of the SOC, SOT, SOD, EOC, and SIZ markers and marker segments, the marker segments can appear in any order within the respective headers.

A.4 Delimiting markers and marker segments

The delimiting marker and marker segments shall be present in all codestreams conforming to this Recommendation | International Standard. Each codestream has only one SOC marker, one EOC marker, and at least one tile-part. Each tile-part has one SOT and one SOD marker. The SOC, SOD, and EOC are delimiting markers, not marker segments, and have no explicit length information or other parameters.

A.4.1 Start of codestream (SOC)

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

Usage: Main header. This is the first marker in the codestream. There shall be only one SOC per codestream.

Length: Fixed.

SOC: Marker code.

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, the index of its tile, and the index of its tile-part. The tile-parts of a given tile shall appear in order (see TPsot) in the codestream. However, tile-parts from other tiles may be interleaved in the codestream. Therefore, the tile-parts from a given tile may not appear contiguously in the codestream.

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

Length: Fixed.

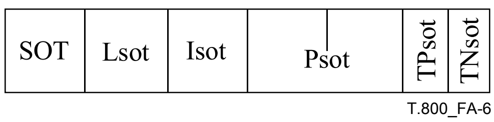


Figure A.6 – Start of tile-part syntax

SOT: Marker code. Table A.5 shows the sizes and values of the symbol and parameters for start of tile-part marker segment.

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

Isot: Tile index. 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.16 shows this alignment. Only the last tile-part in the codestream may contain a 0 for Psot. If the Psot is 0, this tile-part is assumed to contain all data until the EOC marker.

TPsot: Tile-part index. 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. The tile-parts of this tile shall appear in the codestream in this order, although not necessarily consecutively.

TNsot: Number of tile-parts of a tile in the codestream. Two values are allowed: the correct number of tile-parts for that tile and zero. A zero value indicates that the number of tile-parts of this tile is not specified in this tile-part.

Table A.5 – Start of tile-part parameter values

Parameter	Size (bits)	Values
SOT	16	0xFF90
Lsot	16	10
Isot	16	0 to 65 534
Psot	32	0, or 14 to $(2^{32} - 1)$
TPsot	8	0 to 254
TNsot	8	Table A.6

Table A.6 – Number of tile-parts, TNsot, parameter value

Value	Number of tile-parts
0	Number of tile-parts of this tile in the codestream is not defined in this header
1 to 255	Number of tile-parts of this tile in the codestream

A.4.3 Start of data (SOD)

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

Usage: Every tile-part header. Shall be the last marker in a tile-part header. Bit-stream data between an SOD and the next SOT or EOC (end of image) shall be a multiple of 8 bits – the codestream is padded with bits, as needed. There shall be at least one SOD in a codestream. There shall be one SOD per tile-part.

Length: Fixed.

SOD: Marker code

Table A.7 – Start of data parameter values

Parameter	Size (bits)	Values
SOD	16	0xFF93

A.4.4 End of codestream (EOC)

Function: Indicates the end of the codestream.

NOTE 1 – This marker shares the same code 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 EOC per codestream.

NOTE 2 – In the case a file has been corrupted, it is possible that a decoder could extract much useful compressed image data without encountering an EOC marker.

Length: Fixed.

EOC: Marker code

Table A.8 – End of codestream parameter values

Parameter	Size (bits)	Values
EOC	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 bit depth, and the separation of component samples with respect to the reference grid (see B.2).

Usage: Main header. There 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.

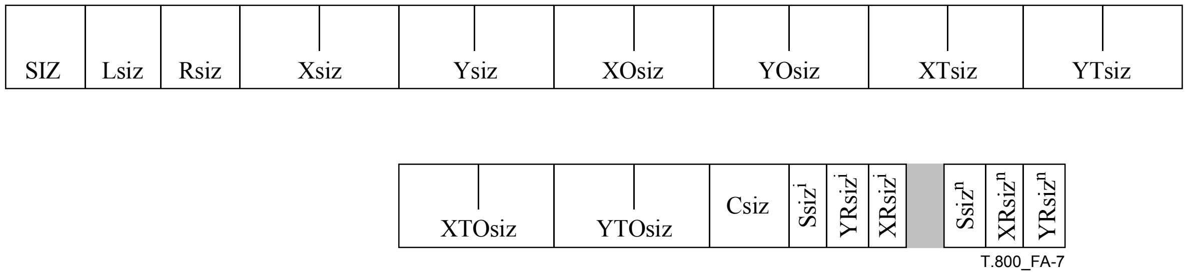


Figure A.7 – Image and tile size syntax

SIZ: Marker code. Table A.9 shows the size and parameter values of the symbol and parameters for image and tile size marker segment.

Lsiz: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

$$Lsiz = 38 + 3 \cdot Csiz \qquad \text{(A-1)}$$

Rsiz: Denotes capabilities that a decoder needs to properly decode the codestream.

Xsiz: Width of the reference grid.

Ysiz: Height of the reference grid.

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.

$Ssiz^i$: Precision (depth) in bits and sign of the ith component samples. The precision is the precision of the component samples before DC level shifting is performed (i.e., the precision of the original component samples before any processing is performed). If the component sample values are signed, then the range of component sample values is $-2^{(Ssiz+1 \text{ AND } 0x7F)-1} \leq$ component sample value $\leq 2^{(Ssiz+1 \text{ AND } 0x7F)-1} - 1$. There is one occurrence of this parameter for each component. The order corresponds to the component's index, starting with zero.

$XRsiz^i$: Horizontal separation of a sample of ith component with respect to the reference grid. There is one occurrence of this parameter for each component.

$YRsiz^i$: Vertical separation of a sample of ith component with respect to the reference grid. There is one occurrence of this parameter for each component.

Table A.9 – Image and tile size parameter values

Parameter	Size (bits)	Values
SIZ	16	0xFF51
Lsiz	16	41 to 49 190
Rsiz	16	Table A.10
Xsiz	32	1 to $(2^{32} – 1)$
Ysiz	32	1 to $(2^{32} – 1)$
XOsiz	32	0 to $(2^{32} – 2)$
YOsiz	32	0 to $(2^{32} – 2)$
XTsiz	32	1 to $(2^{32} – 1)$
YTsiz	32	1 to $(2^{32} – 1)$
XTOsiz	32	0 to $(2^{32} – 2)$
YTOsiz	32	0 to $(2^{32} – 2)$
Csiz	16	1 to 16 384
$Ssiz^i$	8	Table A.11
$XRsiz^i$	8	1 to 255
$YRsiz^i$	8	1 to 255

Table A.10 – Capability Rsiz parameter

Value (bits) MSB LSB	Capability
`0000 0000 0000 0000`	Capabilities specified in this Recommendation | International Standard only
`0000 0000 0000 0001`	Codestream restricted as described for Profile 0 from Table A.45
`0000 0000 0000 0010`	Codestream restricted as described for Profile 1 from Table A.45
	All other values reserved

Table A.11 – Component Ssiz parameter

Value (bits) MSB LSB	Component sample precision

`x000 0000`
to
`x010 0101`
	Component sample bit depth = value + 1. From 1 bit deep through 38 bits deep respectively (counting the sign bit, if appropriate)[a)], R_I

`0xxx xxxx`	Component sample values are unsigned values
`1xxx xxxx`	Component sample values are signed values
	All other values reserved

a) The component sample precision is limited by the number of guard bits, quantization, growth of coefficients at each decomposition level and the number of coding passes that can be signalled. Not all combinations of coding styles will allow the coding of 38-bit samples.

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.

A.6.1 Coding style default (COD)

Function: Describes the coding style, number of decomposition levels, and layering that is the default 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: Main and first tile-part header of a given tile. Shall be one and only one in the main header. Additionally, there may be at most one for each tile. 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 (TPsot = 0).

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 order of precedence is the following:

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

where the "greater than" sign, >, means that the greater overrides the lessor marker segment.

Length: Variable depending on the value of Scod.

Figure A.8 – Coding style default syntax

COD: Marker code. Table A.12 shows the size and values of the symbol and parameters for coding style, default marker segment.

Lcod: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

$$Lcod = \begin{cases} 12 & \text{maximum\_precincts} \\ 13 + \text{number\_decomposition levels} & \text{user-defined\_precincts} \end{cases} \quad \text{(A-2)}$$

where maximum_precincts and user-defined_precincts are indicated in the Scod parameter and number_decomposition_levels are indicated in the SPcod parameter.

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

SGcod: Parameters for coding style designated in Scod. The parameters are independent of components and are designated, in order from top to bottom, in Table A.14. The coding style parameters within the SGcod field appear in the sequence shown in Figure A.9.

SPcod: Parameters for coding style designated in Scod. The parameters relate to all components and are designated, in order from top to bottom, in Table A.15. The coding style parameters within the SPcod field appear in the sequence shown in Figure A.9.

Table A.12 – Coding style default parameter values

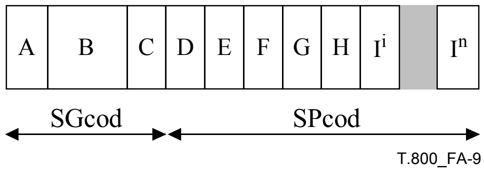
Parameter	Size (bits)	Values
COD	16	0xFF52
Lcod	16	12 to 45
Scod	8	Table A.13
SGcod	32	Table A.14
SPcod	variable	Table A.15

Table A.13 – Coding style parameter values for the Scod parameter

Value (bits) MSB LSB	Coding style
xxxx xxx0	Entropy coder, precincts with PPx = 15 and PPy = 15
xxxx xxx1	Entropy coder with precincts defined below
xxxx xx0x	No SOP marker segments used
xxxx xx1x	SOP marker segments may be used
xxxx x0xx	No EPH marker used
xxxx x1xx	EPH marker shall be used
	All other values reserved

Table A.14 – Coding style parameter values of the SGcod parameter

Parameters (in order)	Size (bits)	Values	Meaning of SGcod values
Progression order	8	Table A.16	Progression order
Number of layers	16	1 to 65 535	Number of layers
Multiple component transformation	8	Table A.17	Multiple component transformation usage



A — Progression order
B — Number of layers
C — Multiple component transformation
D — Number of decomposition levels
E — Code-block width
F — Code-block height
G — Code-block style
H — Transformation
I^i through I^n — Precinct size

Figure A.9 – Coding style parameter diagram of the SGcod and SPcod parameters

Table A.15 – Coding style parameter values of the SPcod and SPcoc parameters

Parameters (in order)	Size (bits)	Values	Meaning of SPcod values
Number of decomposition levels	8	0 to 32	Number of decomposition levels, N_L, Zero implies no transformation.
Code-block width	8	Table A.18	Code-block width exponent offset value, *xcb*
Code-block height	8	Table A.18	Code-block height exponent offset value, *ycb*
Code-block style	8	Table A.19	Style of the code-block coding passes
Transformation	8	Table A.20	Wavelet transformation used
Precinct size	variable	Table A.21	If Scod or Scoc = xxxx xxx0, this parameter is not presen; otherwise this indicates precinct width and height. The first parameter (8 bits) corresponds to the N_LLL sub-band. Each successive parameter corresponds to each successive resolution level in order.

Table A.16 – Progression order for the SGcod, SPcoc, and Ppoc parameters

Value (bits) MSB LSB	Progression order
0000 0000	Layer-resolution level-component-position progression
0000 0001	Resolution level-layer-component-position progression
0000 0010	Resolution level-position-component-layer progression
0000 0011	Position-component-resolution level-layer progression
0000 0100	Component-position-resolution level-layer progression
	All other values reserved

Table A.17 – Multiple component transformation for the SGcod parameters

Value (bits) MSB LSB	Multiple component transformation type
0000 0000	No multiple component transformation specified
0000 0001	Component transformation used on components 0, 1, 2 for coding efficiency (see G.2). Irreversible component transformation used with the 9-7 irreversible filter. Reversible component transformation used with the 5-3 reversible filter
	All other values reserved

Table A.18 – Width or height exponent of the code-blocks for the SPcod and SPcoc parameters

Value (bits) MSB LSB	Code-block width and height
xxxx 0000 to xxxx 1000	Code-block width and height exponent offset value $xcb = value + 2$ or $ycb = value + 2$. 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 so that $xcb + ycb \leq 12$.
	All other values reserved

Table A.19 – Code-block style for the SPcod and SPcoc parameters

Value (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 causal context Vertically causal context
xxx0 xxxx xxx1 xxxx	No predictable termination Predictable termination
xx0x xxxx xx1x xxxx	No segmentation symbols are used Segmentation symbols are used
	All other values reserved

Table A.20 – Transformation for the SPcod and SPcoc parameters

Value (bits) MSB LSB	Transformation type
`0000 0000`	9-7 irreversible filter
`0000 0001`	5-3 reversible filter
	All other values reserved

Table A.21 – Precinct width and height for the SPcod and SPcoc parameters

Value (bits) MSB LSB	Precinct size

`xxxx 0000`
to
`xxxx 1111`
	4 LSBs are the precinct width exponent, *PPx = value*. This value may only equal zero at the resolution level corresponding to the N_LLL band.

`0000 xxxx`
to
`1111 xxxx`
	4 MSBs are the precinct height exponent *PPy = value*. This value may only equal zero at the resolution level corresponding to the N_LLL band.

A.6.2 Coding style component (COC)

Function: Describes the coding style and number of decomposition levels used for compressing a particular component.

Usage: Main and first tile-part header of a given tile. Optional in both the main and tile-part headers. No more than one per any given 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 (TPsot = 0).

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 order of precedence is the following:

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

where the "greater than" sign, >, means that the greater overrides the lessor marker segment.

Length: Variable depending on the value of Scoc.

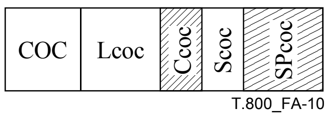


Figure A.10 – Coding style component syntax

COC: Marker code. Table A.22 shows the size and values of the symbol and parameters for coding style component marker segment.

Lcoc: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

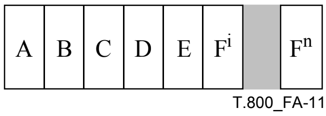
$$Lcoc = \begin{cases} 9 & \text{maximum\_precincts AND Csiz} < 257 \\ 10 & \text{maximum\_precincts AND Csiz} \Rightarrow 257 \\ 10 + \text{number\_decomposition\_levels} & \text{user-defined\_precincts AND Csiz} < 257 \\ 11 + \text{number\_decomposition\_levels} & \text{user-defined\_precincts AND Csiz} \Rightarrow 257 \end{cases} \quad \text{(A-3)}$$

where maximum_precincts and user-defined_precincts are indicated in the Scoc parameter and number_decomposition_levels is indicated in the SPcoc parameter.

Ccoc: The index of the component to which this marker segment relates. The components are indexed 0, 1, 2, etc.

Scoc: Coding style for this component. Table A.23 shows the value for each Scoc parameter.

SPcoc: Parameters for coding style designated in Scoc. The parameters are designated, in order from top to bottom, in Table A.15. The coding style parameters within the SPcoc field appear in the sequence shown in Figure A.11.



A	Number of decomposition levels
B	Code-block width
C	Code-block height
D	Code-block style
E	Transformation
F^i through F^n	Precinct size

Figure A.11 – Coding style parameter diagram of the SPcoc parameters

Table A.22 – Coding style component parameter values

Parameter	Size (bits)	Values
COC	16	0xFF53
Lcoc	16	9 to 43
Ccoc	8 16	0 to 255; if Csiz < 257 0 to 16 383; Csiz ≥ 257
Scoc	8	Table A.23
$SPcoc^i$	variable	Table A.15

Table A.23 – Coding style parameter values for the Scoc parameter

Value (bits) MSB LSB	Coding style
`0000 0000`	Entropy coder with maximum precinct values PPx = PPy = 15
`0000 0001`	Entropy coder with precinct values defined below
	All other values reserved

A.6.3 Region of interest (RGN)

Function: Signals the presence of an ROI in the codestream.

Usage: Main and first tile-part header of a given tile. If used in the main header, it refers to the ROI scaling value for one component in the whole image, valid for all tiles except those with an RGN marker segment.

When used in the tile-part header, the scaling value is valid only for one component in that tile. There may be at most one RGN marker segment for each component in either the main or tile-part headers. The RGN marker segment for a particular component which appears in a tile-part header overrides any marker for that component in the main header, for the tile in which it appears. If there are multiple tile-parts in a tile, then this marker segment shall be found only in the first tile-part header.

Length: Variable.

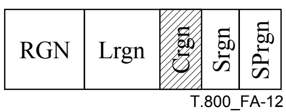


Figure A.12 – Region-of-interest syntax

RGN: Marker code. Table A.24 shows the size and values of the symbol and parameters for region-of-interest marker segment.

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

Crgn: The index of the component to which this marker segment relates. The components are indexed 0, 1, 2, etc.

Srgn: ROI style for the current ROI. Table A.25 shows the value for the Srgn parameter.

SPrgn: Parameter for ROI style designated in Srgn.

Table A.24 – Region-of-interest parameter values

Parameter	Size (bits)	Values
RGN	16	0xFF5E
Lrgn	16	5 to 6
Crgn	8 16	0 to 255; if Csiz < 257 0 to 16 383; Csiz ≥ 257
Srgn	8	Table A.25
SPrgn	8	Table A.26

Table A.25 – Region-of-interest parameter values for the Srgn parameter

Values	ROI style (Srgn)
0	Implicit ROI (maximum shift)
	All other values reserved

Table A.26 – Region-of-interest values from SPrgn parameter (Srgn = 0)

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

A.6.4 Quantization default (QCD)

Function: Describes the quantization default used for compressing all components not defined by a QCC marker segment. The parameter values can be overridden for an individual component by a QCC marker segment in either the main or tile-part header.

Usage: Main and first tile-part header of a given tile. Shall be one and only one in the main header. May be at most one for all tile-part headers of a tile. If there are multiple tile-parts for a tile, and this marker segment is present, it shall be found only in the first tile-part (TPsot = 0).

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

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

where the "greater than" sign, >, means that the greater overrides the lessor marker segment.

Length: Variable depending on the number of quantized elements.

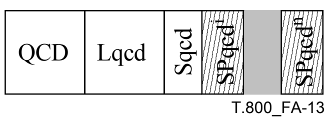


Figure A.13 – Quantization default syntax

QCD: Marker code. Table A.27 shows the size and values of the symbol and parameters for quantization default marker segment.

Lqcd: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

$$Lqcd = \begin{cases} 4 + 3 \cdot \text{number\_decomposition\_levels} & \text{no\_quantization} \\ 5 & \text{scalar\_quantization\_derived} \\ 5 + 6 \cdot \text{number\_decomposition\_levels} & \text{scalar\_quantization\_expounded} \end{cases} \quad \text{(A-4)}$$

where number_decomposition_levels is defined in the COD and COC marker segments, and no_quantization, scalar_quantization_derived, or scalar_quantization_expounded is signalled in the Sqcd parameter.

NOTE – The Lqcd can be used to determine how many quantization step sizes are present in the marker segment. However, there is not necessarily a correspondence with the number of sub-bands present because the sub-bands can be truncated with no requirement to correct this marker segment.

Sqcd: Quantization style for all components.

$SPqcd^i$: Quantization step size value for the ith sub-band in the defined order (see F.3.1). The number of parameters is the same as the number of sub-bands in the tile-component with the greatest number of decomposition levels.

Table A.27 – Quantization default parameter values

Parameter	Size (bits)	Values
QCD	16	0xFF5C
Lqcd	16	4 to 197
Sqcd	8	Table A.28
$SPgcd^i$	variable	Table A.28

Table A.28 – Quantization default values for the Sqcd and Sqcc parameters

Value (bits) MSB LSB	Quantization style	SPqcd or SPqcc size (bits)	SPqcd or SPqcc usage
`xxx0 0000`	No quantization	8	Table A.29
`xxx0 0001`	Scalar derived (values signalled for N_LLL sub-band only). Use Equation (E-5)	16	Table A.30
`xxx0 0010`	Scalar expounded (values signalled for each sub-band). There are as many step sizes signalled as there are sub-bands	16	Table A.30
`000x xxxx` to `111x xxxx`	Number of guard bits: 0 to 7		
	All other values reserved		

Table A.29 – Reversible step size values for the SPqcd and SPqcc parameters (reversible transform only)

Value (bits) MSB LSB	Reversible step size values
`0000 0xxx` to `1111 1xxx`	Exponent, ε_b, of the reversible dynamic range signalled for each sub-band (see Equation (E-5))
	All other values reserved

Table A.30 – Quantization values for the SPqcd and SPqcc parameters (irreversible transformation only)

Value (bits) MSB LSB	Quantization step size values

`xxxx x000 0000 0000`
to
`xxxx x111 1111 1111`
	Mantissa, μ_b, of the quantization step size value (see Equation (E-3))

`0000 0xxx xxxx xxxx`
to
`1111 1xxx xxxx xxxx`
	Exponent, ε_b, of the quantization step size value (see Equation (E-3))

A.6.5 Quantization component (QCC)

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

Usage: Main and first tile-part header of a given tile. Optional in both the main and tile-part headers. No more than one per any given 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 (TPsot = 0).

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 order of precedence is the following:

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

where the "greater than" sign, >, means that the greater overrides the lessor marker segment.

Length: Variable depending on the number of quantized elements.

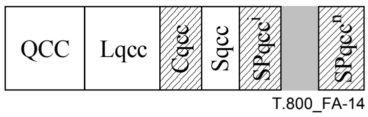


Figure A.14 – Quantization component syntax

QCC: Marker code. Table A.31 shows the size and values of the symbol and parameters for quantization component marker segment.

Lqcc: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

$$Lqcc = \begin{cases} 5 + 3 \cdot \text{number\_decomposition\_levels} & \text{no\_quantization AND Csiz} < 257 \\ 6 & \text{scalar\_quantization\_derived AND Csiz} < 257 \\ 6 + 6 \cdot \text{number\_decomposition\_levels} & \text{scalar\_quantization\_expounded AND Csiz} < 257 \\ 6 + 3 \cdot \text{number\_decomposition\_levels} & \text{no\_quantization AND Csiz} \Rightarrow 257 \\ 7 & \text{scalar\_quantization\_derived AND Csiz} \Rightarrow 257 \\ 7 + 6 \cdot \text{number\_decomposition\_levels} & \text{scalar\_quantization\_expounded AND Csiz} \Rightarrow 257 \end{cases} \quad \text{(A-5)}$$

where number_decomposition_levels is defined in the COD and COC marker segments, and no_quantization, scalar_quantization_derived, or scalar_quantization_expounded is signalled in the Sqcc parameter.

NOTE – The Lqcc can be used to determine how many step sizes are present in the marker segment. However, there is not necessarily a correspondence with the number of sub-bands present because the sub-bands can be truncated with no requirement to correct this marker segment.

Cqcc: The index of the component to which this marker segment relates. The components are indexed 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 F.3.1). The number of parameters is the same as the number of sub-bands in the tile-component with the greatest number of decomposition levels.

Table A.31 – Quantization component parameter values

Parameter	Size (bits)	Values
QCC	16	0xFF5D
Lqcc	16	5 to 199
Cqcc	8 16	0 to 255; if Csiz < 257 0 to 16 383; Csiz ≥ 257
Sqcc	8	Table A.28
$SPqcc^i$	variable	Table A.28

A.6.6 Progression order change (POC)

Function: Describes the bounds and progression order for any progression order other than specified in the COD marker segments in the codestream.

Usage: Main and tile-part headers. At most one POC marker segment may appear in any header. However, several progressions can be described with one POC marker segment. If a POC marker segment is used in the main header, it overrides the progression order in the main and tile COD marker segments. If a POC is used to describe the progression of a particular tile, a POC marker segment must appear in the first tile-part header of that tile. Thus, the progression order of a given tile is determined by the presence of the POC or the values of the COD in the following order of precedence:

Tile-part POC > Main POC > Tile-part COD > Main COD

where the "greater than" sign, >, means that the greater overrides the lessor marker segment.

In the case where a POC marker segment is used, the progression of every packet in the codestream (or for that tile of the codestream) shall be defined in one or more POC marker segments. Each progression order is described in only one POC marker segment and shall be described in any tile-part header before any packets of that progression are found.

Length: Variable depending on the number of different progressions.

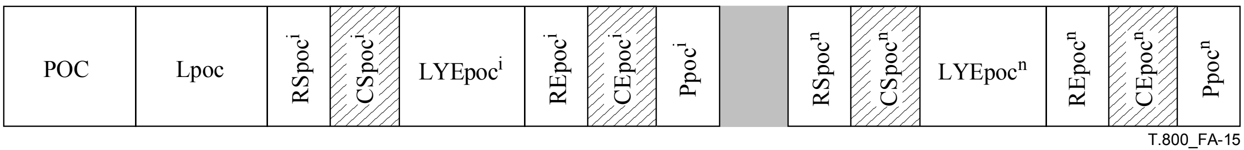


Figure A.15 – Progression order change tile syntax

POC: Marker value. Table A.32 shows the size and values of the symbol and parameters for progression order change marker segment.

Lpoc: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

$$Lpoc = \begin{cases} 2 + 7 \cdot \text{number\_progression\_order\_change} & \text{Csiz} < 257 \\ 2 + 9 \cdot \text{number\_progression\_order\_change} & \text{Csiz} \Rightarrow 257 \end{cases} \tag{A-6}$$

where the number_progression_order_changes is encoder defined.

$RSpoc^i$: Resolution level index (inclusive) for the start of a progression. One value for each progression change in this tile or tile-part. The number of progression changes can be derived from the length of the marker segment.

$CSpoc^i$: Component index (inclusive) for the start of a progression. The components are indexed 0, 1, 2, etc. (either 8 or 16 bits depending on Csiz value). One value for each progression change in this tile or tile-part. The number of progression changes can be derived from the length of the marker segment.

LYEpoc^i: Layer index (exclusive) for the end of a progression. The layer index always starts at zero for every progression. Packets that have already been included in the codestream are not included again. One value for each progression change in this tile or tile-part. The number of progression changes can be derived from the length of the marker segment.

REpoc^i: Resolution Level index (exclusive) for the end of a progression. One value for each progression change in this tile or tile-part. The number of progression changes can be derived from the length of the marker segment.

CEpoc^i: Component index (exclusive) for the end of a progression. The components are indexed 0, 1, 2, etc. (either 8 or 16 bits depending on Csiz value). One value for each progression change in this tile or tile-part. The number of progression changes can be derived from the length of the marker segment.

Ppoc^i: Progression order. One value for each progression change in this tile or tile-part. The number of progression changes can be derived from the length of the marker segment.

Table A.32 – Progression order change, tile parameter values

Parameter	Size (bits)	Values
POC	16	0xFF5F
Lpoc	16	9 to 65 535
RSpoc^i	8	0 to 32
CSpoc^i	8 16	0 to 255; if Csiz < 257 0 to 16 383; Csiz ≥ 257
LYEpoc^i	16	1 to 65 535
REpoc^i	8	(RSpoc^i + 1) to 33
CEpoc^i	8 16	(CSpoc^i + 1) to 255, 0; if Csiz < 257 (CSpoc^i + 1) to 16 384, 0; Csiz ≥ 257 (0 is interpreted as 256)
Ppoc^i	8	Table A.16

A.7 Pointer marker segments

Pointer marker segments either provide a length or pointer 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.

NOTE – Having the pointer marker segments all occur in the main header allows direct access into the bit-stream 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 bit-stream 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.16 shows the length of a tile-part.

The PLM or PLT marker segments are optional. The PLM marker segment is used in the main header and the PLT marker segments are used in tile-part headers. The PLM and PLT marker segments describe the lengths of each packet in the codestream.

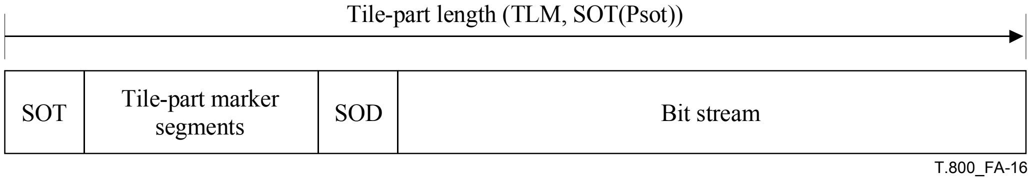


Figure A.16 – Tile-part lengths

A.7.1 Tile-part lengths (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 bit-stream 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: Main header. Optional use in the main header only. There may be multiple TLM marker segments in the main header.

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

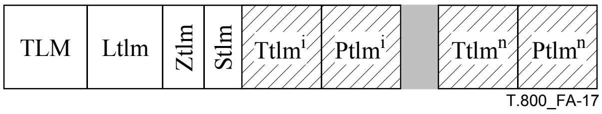


Figure A.17 – Tile part length syntax

TLM: Marker code. Table A.33 shows the size and values of the symbol and parameters for the tile-part length marker segment.

Ltlm: Length of marker segment in bytes (not including the marker). The value of this parameter is determined by the following equation:

$$Ltlm = \begin{cases} 4 + 2 \cdot \text{number\_of\_tile\_parts\_in\_marker\_segment} & \text{ST} = 0 \text{ AND } \text{SP} = 0 \\ 4 + 3 \cdot \text{number\_of\_tile\_parts\_in\_marker\_segment} & \text{ST} = 1 \text{ AND } \text{SP} = 0 \\ 4 + 4 \cdot \text{number\_of\_tile\_parts\_in\_marker\_segment} & \text{ST} = 2 \text{ AND } \text{SP} = 0 \\ 4 + 4 \cdot \text{number\_of\_tile\_parts\_in\_marker\_segment} & \text{ST} = 0 \text{ AND } \text{SP} = 1 \\ 4 + 5 \cdot \text{number\_of\_tile\_parts\_in\_marker\_segment} & \text{ST} = 1 \text{ AND } \text{SP} = 1 \\ 4 + 6 \cdot \text{number\_of\_tile\_parts\_in\_marker\_segment} & \text{ST} = 2 \text{ AND } \text{SP} = 1 \end{cases} \quad \text{(A-7)}$$

where number_of_tile-parts_in_marker_segment is the number of tile-part lengths that are denoted in this marker segment; *ST* and *SP* are signalled by Stlm parameter.

Ztlm: Index of this marker segment relative to all other TLM marker segments present in the current header. The sequence of ($Ttlm^i$, $Ptlm^i$) pairs from this marker segment is concatenated, in order of increasing Ztlm, with the sequences of pairs from other marker segments. The jth entry in the resulting list contains the tile index and tile-part length pair for the jth tile-part appearing in the codestream.

Stlm: Size of the Ttlm and Ptlm parameters.

$Ttlm^i$: Tile index of the ith tile-part. Either none or one value for every tile-part. The number of tile-parts in each tile can be derived from this marker segment (or the concatenated list of all such markers) or from a non-zero TNsot parameter, if present.

$Ptlm^i$: Length, in bytes, from the beginning of the SOT marker of the ith tile-part to the end of the bit stream data for that tile-part. One value for every tile-part.

Table A.33 – Tile-part length parameter values

Parameter	Size (bits)	Values
TLM	16	0xFF55
Ltlm	16	6 to 65 535
Ztlm	8	0 to 255
Stlm	8	Table A.34
$Ttlm^i$	0 if ST = 0 8 if ST = 1 16 if ST = 2	tiles in order 0 to 254 0 to 65 534
$Ptlm^i$	16 if SP = 0 32 if SP = 1	14 to 65 535 14 to (2^{32}– 1)

Table A.34 – Size parameters for Stlm

Value (bits) MSB LSB	Parameter size
xx00 xxxx	ST = 0; Ttlm parameter is 0 bits, only one tile-part per tile and the tiles 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 for every tile-part in order.

Usage: Main header. There may be multiple PLM marker segments. Both the PLM and PLT marker segments are optional and can be used together or separately.

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

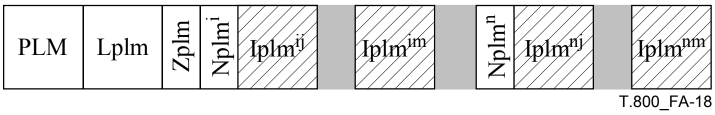


Figure A.18 – Packets length, main header syntax

PLM: Marker code. Table A.35 shows the size and values of the symbol and parameters for the packet length, main header marker segment.

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

Zplm: Index of this marker segment relative to all other PLM marker segments present in the current header. The sequence of ($Nplm^i$, $Iplm^i$) parameters from this marker segment is concatenated, in order of increasing Zplm, with the sequences of parameters from other marker segments. The kth entry in the resulting list contains the number of bytes and packet header pair for the kth tile-part appearing in the codestream.

Every marker segment in this series shall end with a completed packet header length. However, the series of Iplm parameters described by the Nplm does not have to be complete in a given marker segment. Therefore, it is possible that the next PLM marker segment will not have a Nplm parameter after Zplm, but the continuation of the Iplm series from the last PLM marker segment.

$Nplm^i$: Number of bytes of Iplm information for the ith tile-part in the order found in the codestream. One value for each tile-part. If a codestream contains one, or more, tile-parts exceeding the limitations of PLM markers, these markers shall not be used.

NOTE – This value is expressed with an 8-bit number limiting the number of Iplm bytes to 255 and the number of packets in a tile-part to 255, or less. This is not a restriction on the number of packets that can be in a tile-part. It is merely a limit on this marker segment's ability to describe the packets in a tile-part.

$Iplm^{ij}$: Length of the jth packet in the ith tile-part. 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 range of values for each tile-part. One value for each packet in the tile.

Table A.35 – Packets length, main header parameter values

Parameter	Size (bits)	Values
PLM	16	0xFF57
Lplm	16	4 to 65 535
Zplm	8	0 to 255
$Nplm^i$	8	0 to 255
$Iplm^{ij}$	variable	Table A.36

Table A.36 – Iplm, Iplt list of packet lengths

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 to x111 1111	Last 7 bits of packet length, terminate number[a] Continue reading[b] 7 bits of packet length

a) These are the last 7 bits that make up the packet length.

b) These are not the last 7 bits that make up the packet length. Instead, these 7 bits are a portion of those that make up the packet length. The packet length has been broken into 7-bit segments which are sent in order from the most significant segment to the least significant segment. Furthermore, the bits in the most significant segment are right justified to the byte boundary. For example, a packet length of 128 is signalled as 1000 0001 0000 0000, while a length of 512 is signalled as 1000 0100 0000 0000.

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

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

Usage: Tile-part headers. There may be multiple PLT marker segments per tile. Both the PLM and PLT marker segments are optional and can be used together or separately. Shall appear in any tile-part header before the packets whose lengths are described herein.

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

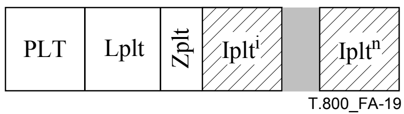


Figure A.19 – Packet length, tile-part header syntax

PLT: Marker code. Table A.37 shows the size and values of the symbol and parameters for the packet length, tile-part header marker segment.

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

Zplt: Index of this marker segment relative to all other PLT marker segments present in the current header. The sequence of ($Iplt^i$) parameters from this marker segment is concatenated, in order of increasing Zplt, with the sequences of parameters from other marker segments. Every marker segment in this series shall end with a completed packet header length.

$Iplm^i$: 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 lengths.

Table A.37 – Packet length, tile-part headers parameter values

Parameter	Size (bits)	Values
PLT	16	0xFF58
Lplt	16	4 to 65 535
Zplt	8	0 to 255
$Iplt^i$	variable	Table A.36

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

Function: A collection of the packet headers from all tiles.

NOTE – This is useful so multiple reads are not required to decode headers.

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

The packet headers shall be in only one of three places within the codestream. If the PPM marker segment is present, all the packet headers shall be found in the main header. 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 then the packet headers can be distributed either in PPT marker segments or distributed in the codestream as defined in B.10. The packet headers shall not be in both a PPT marker segment and the codestream for the same tile. If the packet headers are in PPT marker segments, they shall appear in a tile-part header before the corresponding packet data appears (i.e., in the same tile-part header or one with a lower TPsot value). There may be multiple PPT marker segments in a tile-part header.

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

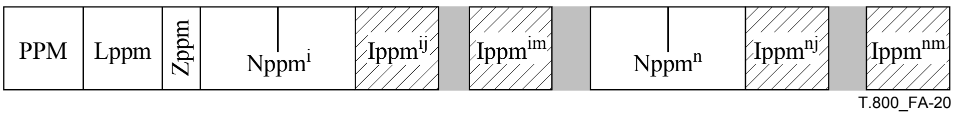


Figure A.20 – Packed packet headers, main header syntax

PPM: Marker code. Table A.38 shows the size and values of the symbol and parameters for the packed packet headers, main header marker segment.

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

Zppm: Index of this marker segment relative to all other PPM marker segments present in the main header. The sequence of ($Nppm^i$, $Ippm^i$) parameters from this marker segment is concatenated, in order of increasing Zppm, with the sequences of parameters from other marker segments. The kth entry in the resulting list contains the number of bytes and packet headers for the kth tile-part appearing in the codestream.

Every marker segment in this series shall end with a completed packet header. However, the series of Ippm parameters described by the Nppm does not have to be complete in a given marker segment. Therefore, it is possible that the next PPM marker segment will not have a Nppm parameter after Zppm, but the continuation of the Ippm series from the last PPM marker segment.

$Nppm^i$: Number of bytes of Ippm information for the ith tile-part in the order found in the codestream. One value for each tile-part (not tile).

$Ippm^{ij}$: Packet header for every packet in order in the tile-part. The contents are exactly the packet header which would have been distributed in the bit stream as described in B.10.

Table A.38 – Packed packet headers, main header parameter values

Parameter	Size (bits)	Values
PPM	16	0xFF60
Lppm	16	7 to 65 535
Zppm	8	0 to 255
$Nppm^i$	32	0 to (2^{32}– 1)
$Ippm^{ij}$	variable	packet headers

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

Function: A collection of the packet headers from one tile or tile-part.

Usage: Tile-part headers. Shall appear in any tile-part header before the packets whose headers are described herein.

The packet headers shall be in only one of three places within the codestream. If the PPM marker segment is present, all the packet headers shall be found in the main header. 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, then the packet headers can be distributed either in PPT marker segments or distributed in the codestream as defined in B.10. The packet headers shall not be in both a PPT marker segment and the codestream for the same tile. If the packet headers are in PPT marker segments, they shall appear in a tile-part header before the corresponding packet data appears (i.e., in the same tile-part header or one with a lower TPsot value). There may be multiple PPT marker segments in a tile-part header.

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

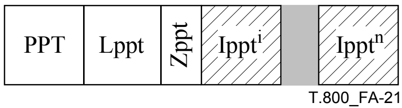


Figure A.21 – Packed packed headers, tile-part header syntax

PPT: Marker code. Table A.39 shows the size and values of the symbol and parameters for the packed packet headers, tile-part header marker segment.

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

Zppt: Index of this marker segment relative to all other PPT marker segments present in the current header. The sequence of ($Ippt^i$) parameters from this marker segment is concatenated, in order of increasing Zppt, with the sequences of parameters from other marker segments. Every marker segment in this series shall end with a completed packet header.

$Ippt^i$: Packet header for every packet in order in the tile-part. The component index, layer, and resolution level are determined from the method of progression or POC marker segments. The contents are exactly the packet header which would have been distributed in the bit stream as described in B.10.

Table A.39 – Packet header, tile-part headers parameter values

Parameter	Size (bits)	Values
PPT	16	0xFF61
Lppt	16	4 to 65 535
Zppt	8	0 to 255
$Ippt^i$	variable	packet headers

A.8 In-bit-stream marker and marker segments

These marker and marker segments are used for error resilience. They can be found in the bit stream. (The EPH marker can also be used in the PPM and PPT marker segments.)

A.8.1 Start of packet (SOP)

Function: Marks the beginning of a packet within a codestream.

Usage: Optional. May be used in the bit stream in front of every packet. Shall not be used unless indicated that it is allowed in the proper COD marker segment (see A.6.1). If PPM or PPT marker segments are used, then the SOP marker segment may appear immediately before the packet data in the bit stream.

If SOP marker segments are allowed (by signalling in the COD marker segment, see A.6.1), each packet in any given tile-part may or may not be appended with an SOP marker segment. However, whether or not the SOP marker segment is used, the count in the Nsop is incremented for each packet. If the packet headers are moved to a PPM or PPT marker segments (see A.7.4 and A.7.5), then the SOP marker segments may appear immediately before the packet body in the tile-part compressed image data portion.

Length: Fixed.

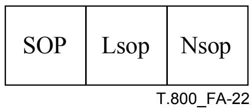


Figure A.22 – Start of packet syntax

SOP: Marker code. Table A.40 shows the size and values of the symbol and parameters for start of packet marker segment.

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

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

Table A.40 – Start of packet parameter values

Parameter	Size (bits)	Values
SOP	16	0xFF91
Lsop	16	4
Nsop	16	0 to 65 535

A.8.2 End of packet header (EPH)

Function: Indicates the end of the packet header for a given packet. This delimits the packet header in the bit stream or in the PPM or PPT marker segments. This marker does not denote the beginning of packet data. If packet headers are not in-bit stream (i.e., PPM or PPT marker segments are used), this marker shall not be used in the bit stream.

Usage: Shall be used if and only if indicated in the proper COD marker segment (see A.6.1). Appears immediately after a packet header.

If EPH markers are required (by signalling in the COD marker segment, see A.6.1), each packet header in any given tile-part shall be postpended with an EPH marker segment. If the packet headers are moved to a PPM or PPT marker segments (see A.7.4 and A.7.5), then the EPH markers shall appear after the packet headers in the PPM or PPT marker segments.

Length: Fixed.

EPH: Marker code

Table A.41 – End of packet header parameter values

Parameter	Size (bits)	Values
EPH	16	0xFF92

A.9 Informational marker segments

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 JP2 (see Annex I).

A.9.1 Component registration (CRG)

Function: Allows specific registration of components with respect to each other. For coding purposes the samples of components are considered to be located at reference grid points that are integer multiples of XRsiz and YRsiz (see A.5.1). However, this may be inappropriate for rendering the image. The CRG marker segment describes the "centre of mass" of each component's samples with respect to the separation. This marker segment has no effect on decoding the codestream.

NOTE – This component registration offset is with respect to the image offset (XOsiz and YOsiz) and the component separation ($XRsiz^i$ and $YRsiz^i$). For example, the horizontal reference grid point for the left-most samples of component *c* is $XRsiz^c \lceil XOsiz / XRsiz^c \rceil$. (Likewise for the vertical direction.) The horizontal offset denoted in this marker segment is in addition to this offset.

Usage: Main header only. Only one CRG may be used in the main header and is applicable for all tiles.

Length: Variable depending on the number of components.

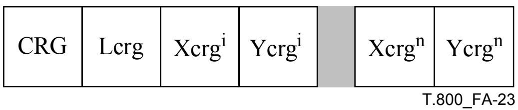


Figure A.23 – Component registration syntax

CRG: Marker code. Table A.42 shows the size and values of the symbol and parameters for the component registration marker segment.

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

$Xcrg^i$: Value of the horizontal offset, in units of 1/65536 of the horizontal separation $XRsiz^i$, for the ith component. Thus, values range from 0/65536 (sample occupies its reference grid point) to $XRsiz^c$(65535/65536) (just before the next sample's reference grid point). This value is repeated for every component.

$Ycrg^i$: Value of the vertical offset, in units of 1/65536 of the vertical separation $YRsiz^i$, for the ith component. Thus, values range from 0/65536 (sample occupies its reference grid point) to $YRsiz^c$(65535/65536) (just before the next sample's reference grid point). This value is repeated for every component.

Table A.42 – Component registration parameter values

Parameter	Size (bits)	Values
CRG	16	0xFF63
Lcrg	16	6 to 65 534
$Xcrg^i$	16	0 to 65 535
$Ycrg^i$	16	0 to 65 535

A.9.2 Comment (COM)

Function: Allows unstructured data in the main and tile-part header.

Usage: Main and tile-part headers. Repeatable as many times as desired in either or both the main or tile-part headers. This marker segment has no effect on decoding the codestream.

Length: Variable depending on the length of the message.

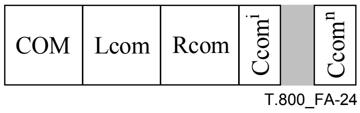


Figure A.24 – Comment syntax

COM: Marker code. Table A.43 shows the size and values of the symbol and parameters for the comment marker segment.

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

Rcom: Registration value of the marker segment.

Ccom^i: Byte of unstructured data.

Table A.43 – Comment parameter values

Parameter	Size (bits)	Values
COM	16	0xFF64
Lcom	16	5 to 65 535
Rcom	16	Table A.44
Ccom^i	8	0 to 255

Table A.44 – Registration values for the Rcom parameter

Values	Registration values
0	General use (binary values)
1	General use (ISO/IEC 8859-15 (Latin) values)
	All other values reserved

A.10 Codestream restrictions conforming to this Recommendation | International Standard

In order to promote the wide inter-operability of JPEG 2000 codestream, codestream restrictions are introduced. "Codestream Restrictions" have two profiles, Profile-0 and Profile-1. The case of "No Restrictions" meaning conforming to this Recommendation | International Standard can be called Profile-2. Profile-0 and Profile-1 are defined as follows.

Maximum interchange will be achieved for codestreams corresponding to Profile-0, and medium interchange for codestreams corresponding to Profile-1.

Table A.45 – Codestream restrictions

Restrictions	Profile-0	Profile-1
SIZ marker segment		
Profile indication	Rsiz = 1	Rsiz = 2
Image size	Xsiz, Ysiz < 2^{31}	Xsiz, Ysiz < 2^{31}
Tiles	Tiles of a dimension 128 × 128: YTsiz = XTsiz = 128 or one tile for the whole image: YTsiz + YTOsiz ≥ Ysiz XTsiz + XTOsiz ≥ Xsiz	XTsiz / min($XRsiz^i$, $YRsiz^i$) ≥ 1024 XTsiz = YTsiz or one tile for the whole image: YTsiz + YTOsiz ≥ Ysiz XTsiz + XTOsiz ≥ Xsiz
Image and tile origin	XOsiz = YOsiz = XTOsiz = YTOsiz = 0	XOsiz, YOsiz, XTOsiz, YTOsiz < 2^{31}
RGN marker segment	SPrgn ≤ 37	SPrgn ≤ 37
Sub-sampling	$XRsiz^i$ = 1, 2, or 4 $YRsiz^i$ = 1, 2, or 4	No restriction
Code-blocks		
Code-block size	*xcb* = *ycb* = 5 or *xcb* = *ycb* = 6	*xcb* ≤ 6, *ycb* ≤ 6
Code-block style	SPcod, SPcoc = 00sp vtra where a = r = v = 0, and t, p, s = 0 or 1 NOTE 1 – t = 1 for termination on each coding pass p = 1 for predictive termination s = 1 for segmentation symbols	No restriction
Marker locations		
Packed headers (PPM, PPT)	Disallowed	No restriction
COD, COC, QCD, QCC	Main header only	No restriction
Subset requirements		
LL resolution	If one tile is used for whole image, (Xsiz – XOsiz) / D(I) ≤ 128 and (Ysiz – YOsiz) / D(I) ≤ 128 where D(I) = $2^{number\_of\_decomposition\_levels}$ in SPcod or SPcoc, for I = component 0 to 3	For each tile in the image, $\lfloor x1 / D(i) \rfloor - \lfloor tx0/D(i) \rfloor$ ≤ 128 and $\lfloor ty1/D(i) \rfloor - \lfloor ty0/D(i) \rfloor$ ≤ 128 where D(I) = $2^{number\_of\_decomposition\_levels}$ in SPcod or SPcoc, for I = component 0 to 3. NOTE 2 – tx0, tx1, ty0 and ty1 are as defined by Equations (B-7) to (B-10).
Parsability	If the POC marker is present, the POC marker shall have RSPOC0 = 0 and CSPOC0 = 0. NOTE 3 – Some compliant decoders might decode only packets associated with the first progression.	No restriction
Tile-parts	Tile-parts with TPsot = 0 of every tile before any tile-parts with TPsot > 0, Tile-parts Isot = 0 to Isot = number_of_tiles – 1, in sequential order for all tile-parts with TPsot = 0	No restriction
Precinct size	"Precinct size" defined by SPcod or SPcoc (Tables A.15 and A.21) must be large enough so there is only one precinct in all resolution levels with dimension less than or equal to 128 by 128. NOTE 4 – Precinct size PPx ≥ 7 and PPy ≥ 7 is sufficient to guarantee only one precinct per sub-band when XOsiz = 0 and YOsiz = 0.	No restriction

Annex B

Image and compressed image data ordering

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

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.

This annex describes the various structural entities, and their organization in the codestream: components, tiles, sub-bands, and their divisions.

B.1 Introduction to image data structure concepts

The reference grid provides a mechanism for co-registering components and for defining subsets of the reference grid, e.g., the image area and tiles.

The components consist of two-dimensional arrays of samples. Each component, *c*, has parameters $XRsiz^c$, $YRsiz^c$ (see A.5.1) which define the mapping between component samples and the reference grid points. Every component sample is associated with a reference grid point (though not vice versa). This mapping induces a registration of components with each other used for coding only.

Each component is divided into tiles corresponding to the tiling of the reference grid. These tile-components are coded independently. Each tile-component is wavelet transformed into several decomposition levels which are related to resolution levels (see Annex F). Each resolution level consists of either the HL, LH, and HH sub-bands from one decomposition level or the N_LLL sub-band. Thus, there is one more resolution level than there are decomposition levels.

Each sub-band has its own origin. The sub-band boundary conditions are unique for each HL, LH, and HH sub-band.

NOTE – This convention differs from the usual wavelet diagrams which place all sub-bands for a component in a single space

Precincts and code-blocks are defined at the resolution level and sub-band. Consequently they can vary over tile-components. Precincts are defined so that code-blocks fit neatly, i.e., they "line up" with each other.

In the accompanying figures, boundaries and coordinate axes are shown. In each case, the samples or coefficients coincident with the left and upper boundaries are included in a given region, while samples or coefficients along the right and/or lower boundaries are not included in that region.

Also, in the accompanying formulae, many of the variables have values that can change as a function of component, tile, or resolution level. These values may change explicitly (through syntax described in Annex A) or implicitly (through propagation). For convenience of notation, some dependencies are suppressed in the discussion that follows.

B.2 Component mapping to the reference grid

All components (and many other structures in this annex) are defined with respect to the reference grid. The various parameters defining the reference grid appear in Figure B.1. The reference grid is a rectangular grid of points with the indices from (0, 0) to (Xsiz – 1, Ysiz – 1). An "image area" is defined on the reference grid by the dimensional parameters, (Xsiz, Ysiz) and (XOsiz, YOsiz). Specifically, the image area on the reference grid is defined by its upper left hand reference grid point at location (XOsiz, YOsiz), and its lower right hand reference grid point at location (Xsiz – 1, Ysiz – 1).

The samples of component *c* are at integer multiples of ($XRsiz^c$, $YRsiz^c$) on the reference grid. Each component domain is a sub-sampled version of the reference grid with the (0, 0) coordinate as common point for each component. Row samples are located reference grid points that are at integer multiples of $XRsiz^c$ and column samples are located reference grid points that are at integer multiples of $YRsiz^c$. Only those samples which fall within the image area actually belong to the image component. Thus, the samples of component *c* are mapped to rectangle having upper left hand sample with coordinates (x_0, y_0) and lower right hand sample with coordinates ($x_1 – 1$, $y_1 – 1$), where:

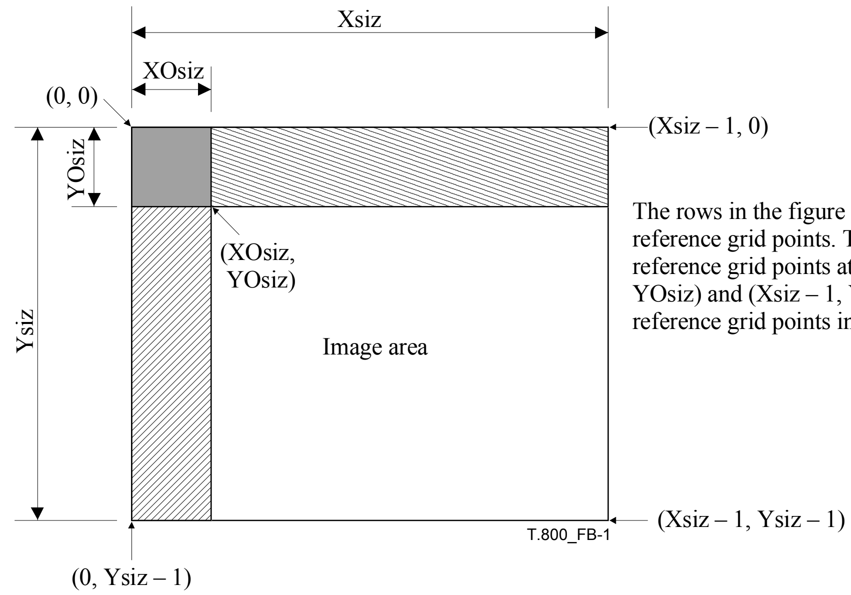


Figure B.1 – Reference grid diagram

$$x_0 = \left\lceil \frac{XOsiz}{XRsiz^c} \right\rceil \quad x_1 = \left\lceil \frac{Xsiz}{XRsiz^c} \right\rceil \quad y_0 = \left\lceil \frac{YOsiz}{YRsiz^c} \right\rceil \quad y_1 = \left\lceil \frac{Ysiz}{YRsiz^c} \right\rceil \qquad \text{(B-1)}$$

Thus, the dimensions of component c are given by:

$$(width, height) = (x_1 - x_0, y_1 - y_0) \qquad \text{(B-2)}$$

The parameters, Xsiz, Ysiz, XOsiz, YOsiz, $XRsiz^c$ and $YRsiz^c$ are all defined in the SIZ marker segment (see A.5.1).

NOTE 1 – The fact that all components share the image offset (XOsiz, YOsiz) and size (Xsiz, Ysiz) induces a registration of the components.

NOTE 2 – Figure B.2 shows an example of three components mapped to the reference grid. Figure B.3 shows the image area from a particular image offset with different (XRsiz, YRsiz) values. The upper left sample coordinate, in the image component domain, that is included in the image area is also illustrated.

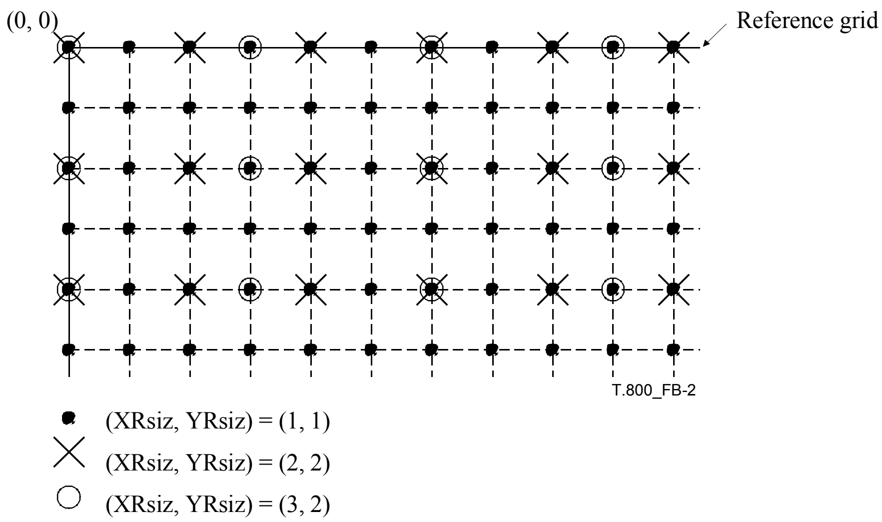


Figure B.2 – Component sample locations on the reference grid for different XRsiz and YRsiz values

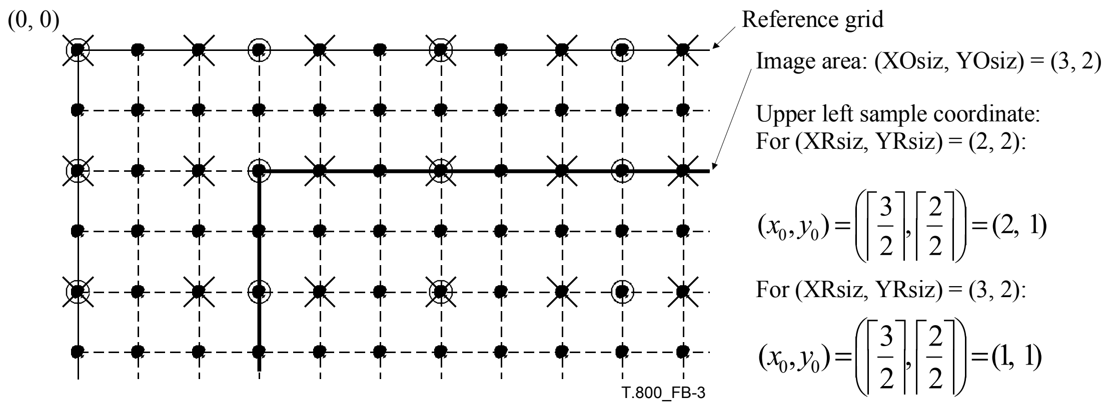


Figure B.3 – Example of upper left component sample locations

B.3 Image area division into tiles and tile-components

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

Every tile is XTsiz reference grid points wide and YTsiz reference grid points high. The top left corner of 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. This is the tile index in the Isot parameter from the SOT marker segment in A.4.2. Thus, the first tile's upper left coordinates relative to the reference grid are (XTOsiz, YTOsiz). Figure B.4 shows this relationship.

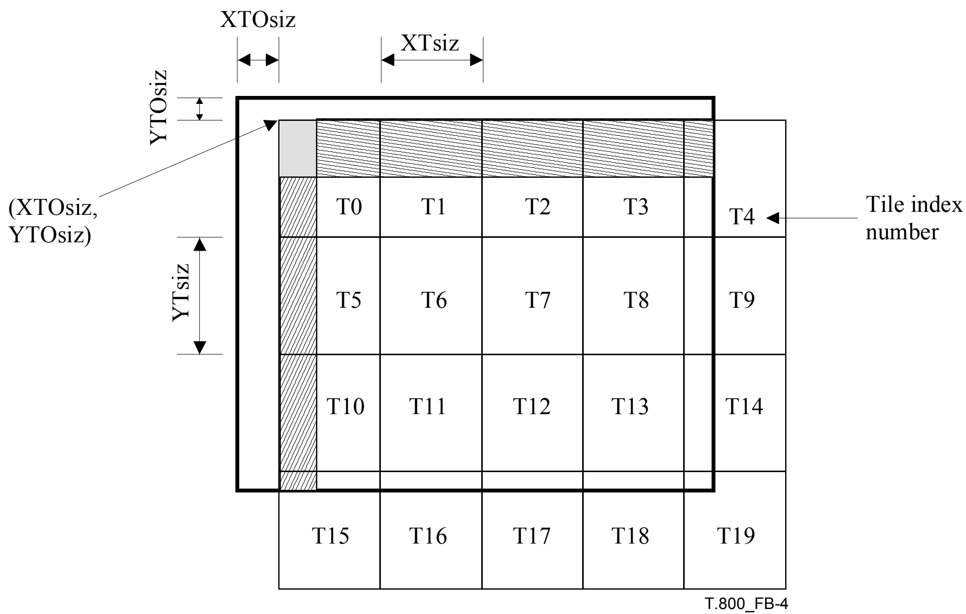


Figure B.4 – Tiling of the reference grid diagram

The tile grid offsets (XTOsiz, YTOsiz) are constrained to be no greater than the image area offsets. This is expressed by the following ranges:

$$0 \le XTOsiz \le XOsiz \qquad 0 \le YTOsiz \le YOsiz \tag{B-3}$$

Also, the tile size plus the tile offset shall be greater than the image area offset. This ensures 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:

$$XTsiz + XTOsiz > XOsiz \qquad YTsiz + YTOsiz > YOsiz \tag{B-4}$$

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

$$numXtiles = \left\lceil \frac{Xsiz - XTOsiz}{XTsiz} \right\rceil \qquad numYtiles = \left\lceil \frac{Ysiz - YTOsiz}{YTsiz} \right\rceil \tag{B-5}$$

For the purposes of this description, it is useful to have tiles indexed in terms of horizontal and vertical position. Let p be the horizontal index of a tile, ranging from 0 to $numXtiles - 1$, and q be the vertical index of a tile, ranging from 0 to $numYtiles - 1$, determined from the tile index as follows:

$$p = mod\ (t, numXtiles) \qquad q = \left\lfloor \frac{t}{numXtiles} \right\rfloor \tag{B-6}$$

where t is the index of the tile in Figure B.4.

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

$$tx_0(p,q) = max(XTOsiz + p \cdot XTsiz,\ XOsiz) \tag{B-7}$$

$$ty_0(p,q) = max(YTOsiz + q \cdot YTsiz, YOsiz) \tag{B-8}$$

$$tx_1(p,q) = min(XTOsiz + (p+1) \cdot XTsiz, Xsiz) \tag{B-9}$$

$$ty_1(p,q) = min(YTOsiz + (q+1) \cdot YTsiz, Ysiz) \tag{B-10}$$

where $tx_0(p, q)$ and $ty_0(p, q)$ are the coordinates of the upper left corner of the tile, $tx_1(p, q) - 1$ and $ty1(p, q) - 1$ are the coordinates of the lower right corner of the tile. We will often drop the tile's coordinates in referring to a specific tile and refer to the coordinates (tx_0, ty_0) and (tx_1, ty_1).

Thus the dimensions of a tile in the reference grid are:

$$(tx_1 - tx_0,\ ty_1 - ty_0) \tag{B-11}$$

Within the domain of image component i, the coordinates of the upper left hand sample are given by (tcx_0, tcy_0) and the coordinates of the lower right hand sample are given by ($tcx_1 - 1$, $tcy_1 - 1$), where:

$$tcx_0 = \left\lceil \frac{tx_0}{XRsiz^i} \right\rceil \quad tcx_1 = \left\lceil \frac{tx_1}{XRsiz^i} \right\rceil \quad tcy_0 = \left\lceil \frac{ty_0}{YRsiz^i} \right\rceil \quad tcy_1 = \left\lceil \frac{ty_1}{YRsiz^i} \right\rceil \tag{B-12}$$

so that the dimensions of the tile-component are:

$$(tcx_1 - tcx_0,\ tcy_1 - tcy_0) \tag{B-13}$$

B.4 Example of the mapping of components to the reference grid (informative)

The following example is included to illustrate the mapping of image components to the reference grid and the area induced by tiling across components with different sub-sampling factors. The example assumes an application in which an original image with aspect ratio 16:9 is to be compressed with this Recommendation | International Standard. Choices of the image size, image offset, tile size, and tile offset are used such that an image with aspect ratio 4:3 can be cropped from the center of the original image. Figure B.5 shows the reference grid and image areas along with the tiling structure that will be imposed in this example.

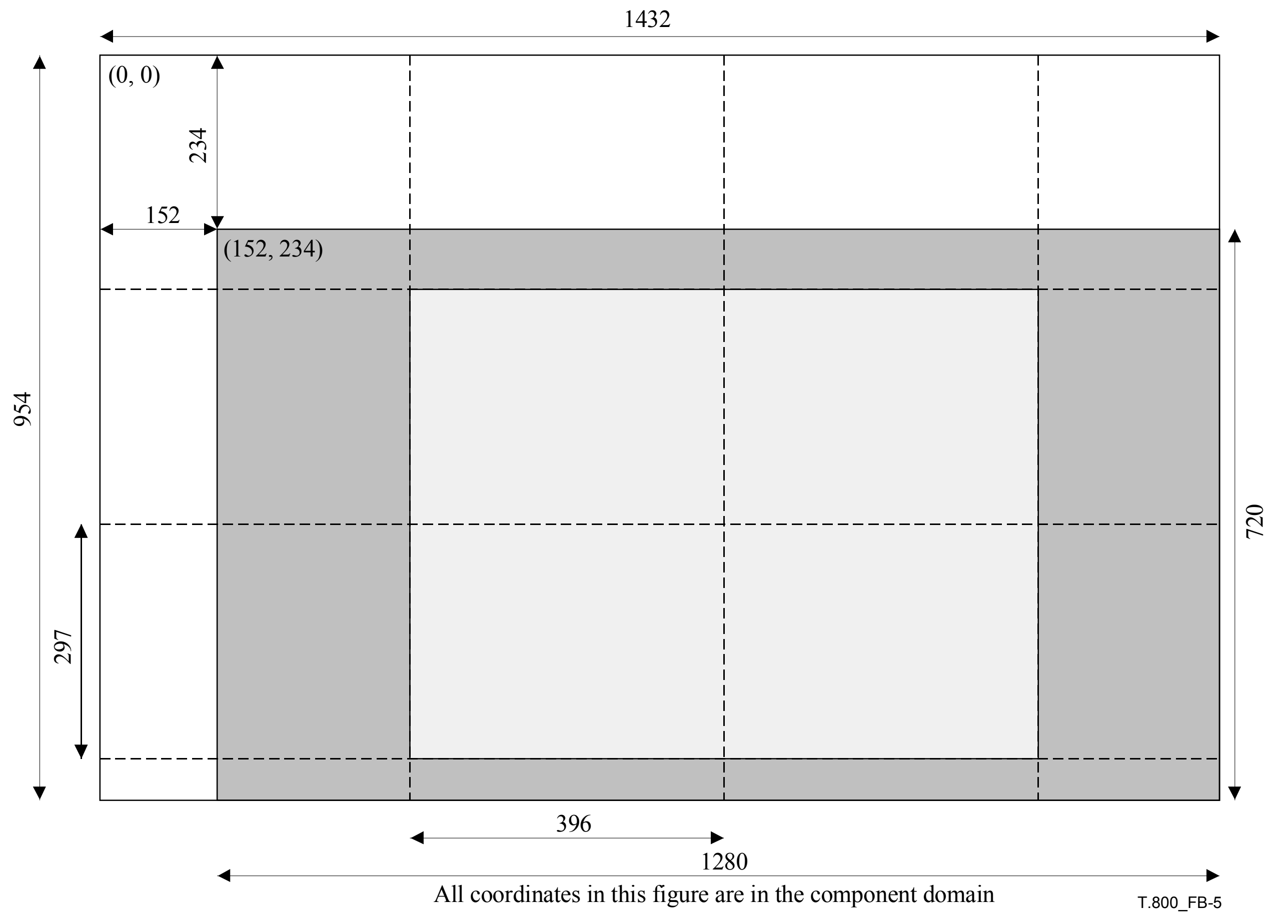


Figure B.5 – Reference grid example

Let the reference grid size (Xsiz, Ysiz) be (1432, 954). In this example, the image will contain two components (component indices will be represented by i = 0, 1). The sub-sampling factors $XRsiz^i$ and $YRsiz^i$ of the two components with respect to the reference grid will be $XRsiz^0 = YRsiz^0 = 1$ and $XRsiz^1 = YRsiz^1 = 2$. The image offset is set to be (XOsiz, YOsiz) = (152, 234). Given these parameters, the sizes of the two image components can be determined from Equation (B-1). The upper left corner of component 0 is found at ($\lceil 152/1 \rceil$, $\lceil 234/1 \rceil$) = (152, 234). The lower right corner of component 0 is found at ($\lceil 1432/1 \rceil - 1$, $\lceil 954/1 \rceil - 1$) = (1431, 953). The actual size of component 0 is therefore 1280 samples in width by 720 samples in height. The upper left corner of component 1 is found at ($\lceil 152/2 \rceil$, $\lceil 234/2 \rceil$) = (76, 117), while the lower right corner of that component is found at ($\lceil 1432/2 \rceil - 1$, $\lceil 954/2 \rceil - 1$) = (715, 476). The actual size of component 1 is therefore 640 samples in width by 360 samples in height.

The tiles are chosen to have an aspect ratio of 4:3. In this example, (XTsiz, YTsiz) will be set to (396, 297) and the tile offsets (XTOsiz, YTOsiz) will be set to (0, 0). The number of tiles in the x and y directions are then determined from Equation (B-5) *numXtiles* = $\lceil 1432/396 \rceil$ = 4, *numYtiles* = $\lceil 954/297 \rceil$ = 4. The tiled image components will therefore contain a total of $t = 16$ tiles, with tile grid indices p and q in the range $0 \le p, q < 4$. It is now possible to compute the locations of the tiles in each image component. To do so, the values of tx_0, tx_1, ty_0, and ty_1 are determined from Equations (B-7), (B-8), (B-9), and (B-10). Since p and q share the same set of admissible values, the notation '0:3' will be used to refer to the sequence of values {0, 1, 2, 3}, and the notation '*' will be used to denote that the result is valid for all admissible values. The values of tx_0 are found as $tx_0(0{:}3, *) = \{152, 396, 792, 1188\}$, and the values of tx_1 are given by $tx_1(0{:}3, *) = \{396, 792, 1188, 1432\}$. The values of ty_0 are $ty_0(*, 0{:}3) = \{234, 297, 594, 891\}$, and the values of ty_1 are $ty_1(*, 0{:}3) = \{297, 594, 891, 954\}$.

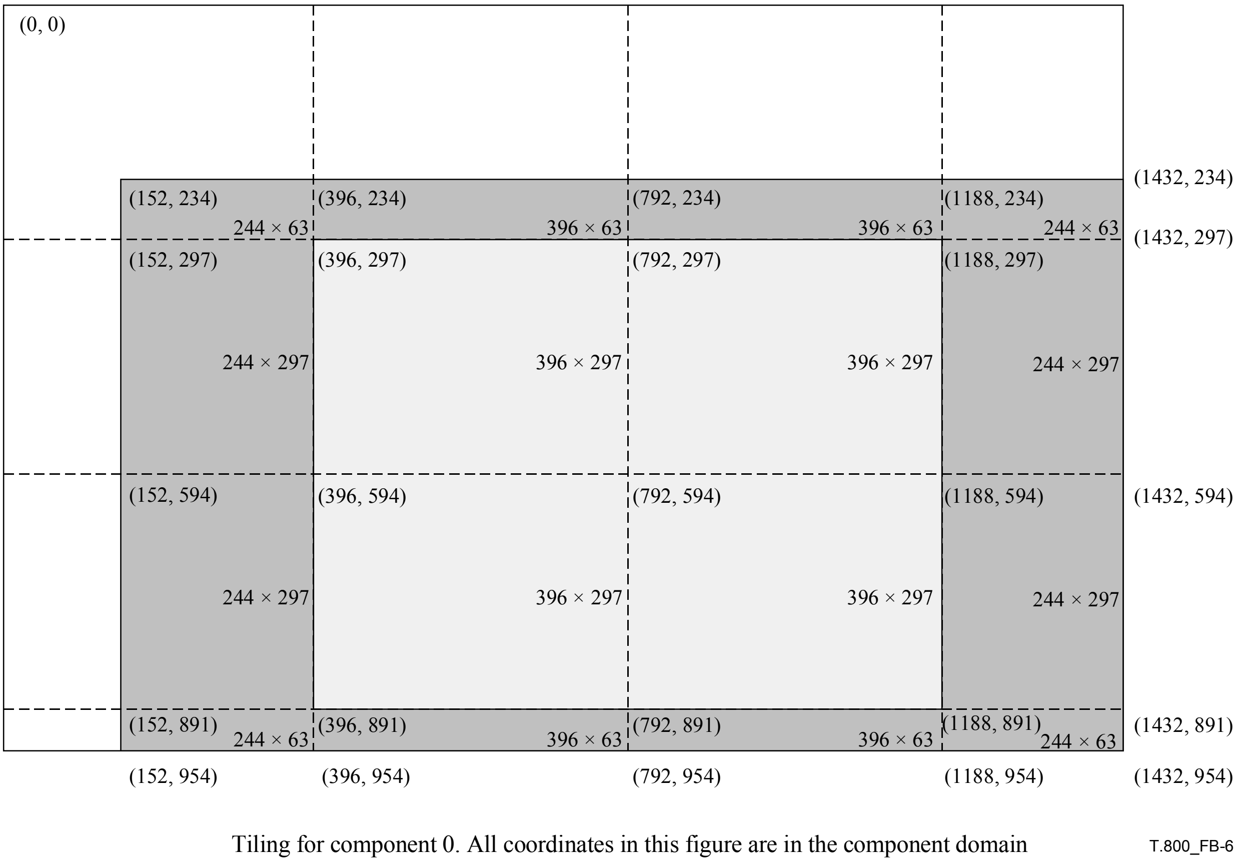
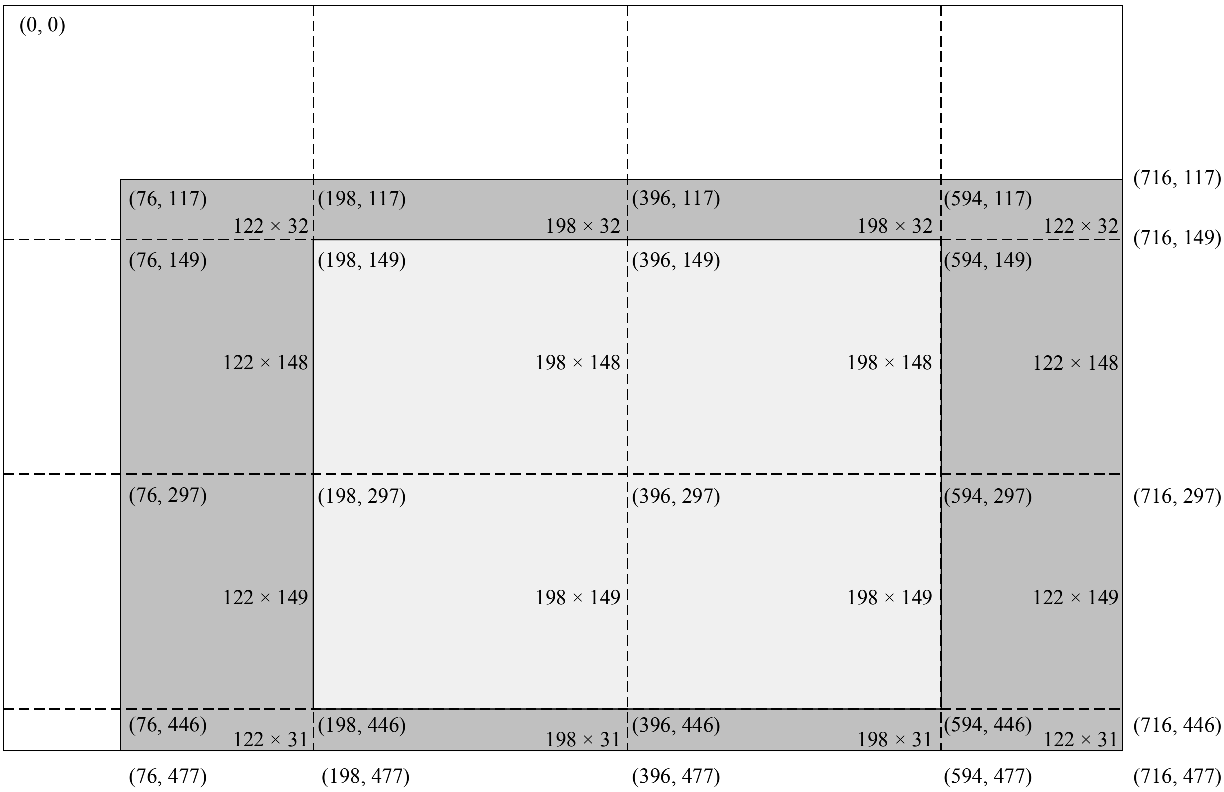


Figure B.6 – Example tile sizes and locations for component 0

With the values of tx_0, tx_1, ty_0, and ty_1 now known, the locations and sizes of all tiles can be determined for each of the components. To do so, Equation (B-12) is used. The relevant locations and sizes for component 0 are shown in Figure B.6, while the same information is provided for component 1 in Figure B.7. Of particular interest are the 'interior' tiles in the figures (tiles (1, 1), (1, 2), (2, 1), and (2, 2)). These tiles are not limited in extent by the image area. In component 0, all of these tiles are the same size. This regularity is a result of the fact that the sub-sampling factors for this component are ($XRsiz^0$, $YRsiz^0$) = (1, 1). However, in component 1, these tiles are not all the same size because ($XRsiz^1$, $YRsiz^1$) = (2, 2). Notice that tiles (1, 1) and (2, 1) are both of size 198 by 148, while tiles (1, 2) and (2, 2) are both of size 198 by 149. This illustrates that the number of samples in the interior tiles of a component can vary depending upon the particular combination of tile size and component sub-sampling factors.

With these choices of reference grid, image offset, tile size, and tile offset, the coded image can be cropped directly to the desired interior region. The four interior tiles from each component can be retained and will represent a cropped image of reference grid size (792, 594). When such a cropping is performed, it will not be necessary to recode the tiles, but the values of some of the reference grid parameters must change. The image offsets must be set to the coordinates of the cropping locations, so that (XOsiz′, YOsiz′) = (396, 297) where (XOsiz′, YOsiz′) are the image offsets of the cropped image. Similarly, the image size must be adjusted to reflect the cropped size (Xsiz′, Ysiz′) = (1188, 891) where (Xsiz′, Ysiz′) are the sizes of the cropped reference grid. Finally, the tile offsets are no longer zero and instead must be set to (XTOsiz′, YTOsiz′) = (396, 297) where (XTOsiz′, YTOsiz′) are the tile offsets of the cropped reference grid.



Tiling for component 1. All coordinates in this figure are in the component domain T.800_FB-7

Figure B.7 – Example tile sizes and locations for component 1

B.5 Transformed tile-component division into resolution levels and sub-bands

Each tile-component is wavelet transformed with N_L decomposition levels as explained in Annex F. Thus, there are $N_L + 1$ distinct resolution levels, denoted $r = 0,1,...,N_L$. The lowest resolution level, $r = 0$, is represented by the N_LLL band. In general, a reduced resolution version of a tile-component with resolution level, r, is the sub-band nLL, where $n = N_L - r$. This clause describes the dimensions of this reduced resolution.

The given tile-component's coordinates with respect to the reference grid at a particular resolution level, r, yield upper left hand sample coordinates, (trx_0, try_0) and lower right hand sample coordinates, $(trx_1 - 1, try_1 - 1)$, where:

$$trx_0 = \left\lceil \frac{tcx_0}{2^{N_L - r}} \right\rceil \quad try_0 = \left\lceil \frac{tcy_0}{2^{N_L - r}} \right\rceil \quad trx_1 = \left\lceil \frac{tcx_1}{2^{N_L - r}} \right\rceil \quad try1 = \left\lceil \frac{tcy_1}{2^{N_L - r}} \right\rceil \qquad \text{(B-14)}$$

In a similar manner, the tile coordinates may be mapped into any particular sub-band, b, yielding upper left hand sample coordinates (tbx_0, tby_0) and lower right hand sample coordinates $(tbx_1 - 1, tby_1 - 1)$ where:

$$\begin{aligned} tbx_0 &= \left\lceil \frac{tcx_0 - \left(2^{n_b - 1} \cdot xo_b\right)}{2^{n_b}} \right\rceil \quad & tby_0 &= \left\lceil \frac{tcy_0 - \left(2^{n_b - 1} \cdot yo_b\right)}{2^{n_b}} \right\rceil \\ tbx_1 &= \left\lceil \frac{tcx_1 - \left(2^{n_b - 1} \cdot xo_b\right)}{2^{n_b}} \right\rceil \quad & tby_1 &= \left\lceil \frac{tcy_1 - \left(2^{n_b - 1} \cdot yo_b\right)}{2^{n_b}} \right\rceil \end{aligned} \qquad \text{(B-15)}$$

where n_b is the decomposition level associated with sub-band b, as discussed in Annex F, and the quantities (xo_b, yo_b) are given by the Table B.1.

Table B.1 – Quantities (xo_b, yo_b) for sub-band b

Sub-band	xo_b	yo_b
n_bLL	0	0
n_bHL (horizontal high-pass)	1	0
n_bLH (vertical high-pass)	0	1
n_bHH	1	1

NOTE – Each of the sub-band is different as mentioned in B.1.

For each sub-band, these coordinates define tile boundaries in distinct sub-band domains. Furthermore, the width of each sub-band within its domain (at the current decomposition level) is given by $tbx_1 - tbx_0$, and the height is given by $tby_1 - tby_0$.

B.6 Division of resolution levels into precincts

Consider a particular tile-component and resolution level whose bounding sample coordinates in the reduced resolution image domain are (trx_0, try_0) and ($trx_1 - 1$, $try_1 - 1$), as already described. Figure B.8 shows the partitioning of this tile-component resolution level into precincts.The precinct is anchored at location (0, 0), so that the upper left hand corner of any given precinct in the partition is located at integer multiples of (2^{PPx}, 2^{PPy}) where *PPx* and *PPy* are signalled in the COD or COC marker segments (see A.6.1 and A.6.2). *PPx* and *PPy* may be different for each tile-component and resolution level. *PPx* and *PPy* must be at least 1 for all resolution levels except $r = 0$ where they are allowed to be zero.

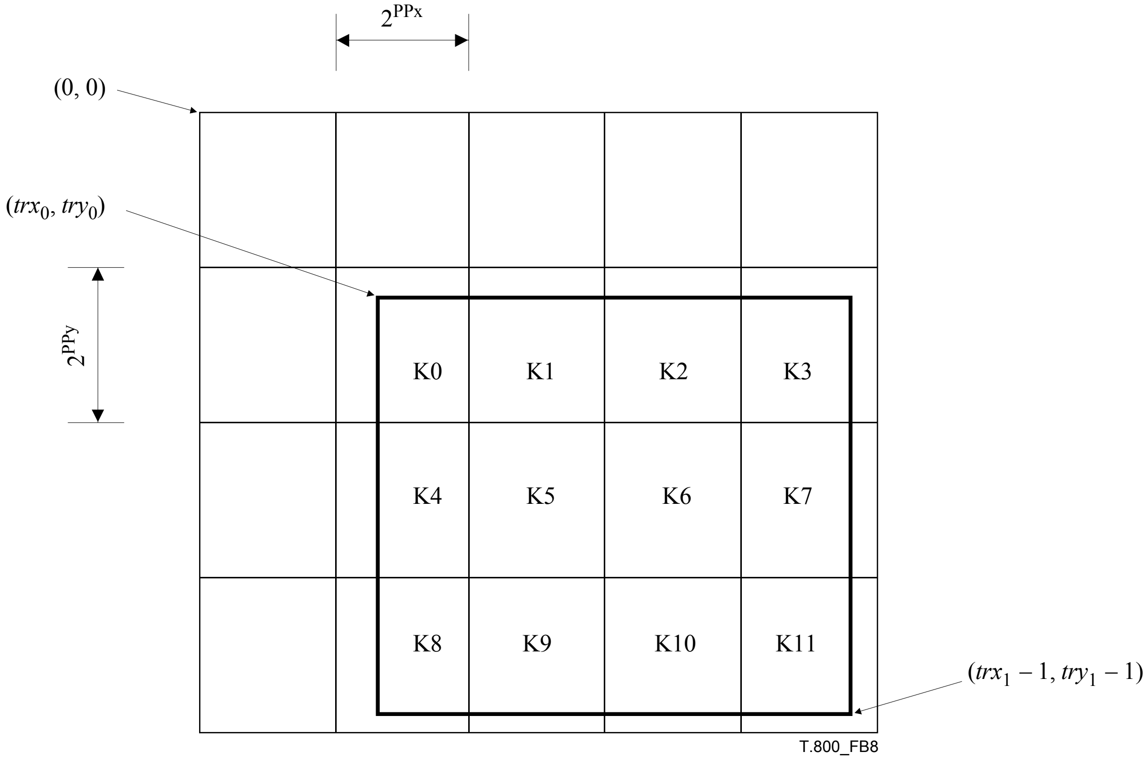


Figure B.8 – Precincts of one reduced resolution

The number of precincts which span the tile-component at resolution level, *r*, is given by:

$$numprecinctswide = \begin{cases} \left\lceil \frac{trx_1}{2^{PPx}} \right\rceil - \left\lfloor \frac{trx_0}{2^{PPx}} \right\rfloor & trx_1 > trx_0 \\ 0 & trx_1 = trx_0 \end{cases} \qquad numprecinctshigh = \begin{cases} \left\lceil \frac{try_1}{2^{PPy}} \right\rceil - \left\lfloor \frac{try_0}{2^{PPy}} \right\rfloor & try_1 > try_0 \\ 0 & try_1 = try_0 \end{cases} \quad \text{(B-16)}$$

Even if Equation (B-16) indicates that both *numprecinctswide* and *numprecinctshigh* are nonzero, some, or all, precincts may still be empty as explained below. The precinct index runs from 0 to *numprecincts* – 1 where *numprecincts* = *numprecinctswide* * *numprecinctshigh* in raster order (see Figure B.8). This index is used in determining the order of appearance, in the codestream, of packets corresponding to each precinct, as explained in B.12.

It can happen that *numprecincts* is 0 for a particular tile-component and resolution level. When this happens, there are no packets for this tile-component and resolution level.

It can happen that a precinct is empty, meaning that no sub-band coefficients from the relevant resolution level actually contribute to the precinct. This can occur, for example, at the lower right of a tile-component due to sampling with respect to the reference grid. When this happens, every packet corresponding to that precinct must still appear in the codestream (see B.9).

B.7 Division of the sub-bands into code-blocks

The sub-bands are partitioned into rectangular code-blocks for the purpose of coefficient modeling and coding. The size of each code-block is determined from two parameters, *xcb* and *ycb*, which are signalled in the COD or COC marker segments (see A.6.1 and A.6.2). The code-block size is the same from all resolution levels. However, at each resolution level, the code-block size is bounded by the precinct size. The code-block size for each sub-band at a particular resolution level is determined as $2^{xcb'}$ by $2^{ycb'}$ where:

$$xcb' = \begin{cases} min(xcb, PPx - 1), \text{ for } r > 0 \\ min(xcb, PPx), \text{ for } r = 0 \end{cases} \tag{B-17}$$

and:

$$ycb' = \begin{cases} min(ycb, PPy - 1), \text{ for } r > 0 \\ min(ycb, PPy), \text{ for } r = 0 \end{cases} \tag{B-18}$$

These equations reflect the fact that the code-block size is constrained both by the precinct size and the code-block size, whose parameters, *xcb* and *ycb*, are identical for all sub-bands in the tile-component. Like the precinct, the code-block partition is anchored at (0, 0), as illustrated in Figure B.9. Thus, all first rows of code-blocks in the code-block partition are located at $y = m2^{ycb'}$ and all first columns of code-blocks are located at $x = n2^{xcb'}$, where *m* and *n* are integers.

NOTE – Code-blocks in the partition may extend beyond the boundaries of the sub-band coefficients. When this happens, only the coefficients lying within the sub-band are coded using the method described in Annex D. The first stripe coded using this method corresponds to the first four rows of sub-band coefficients in the code-block or as many of such rows as are present.

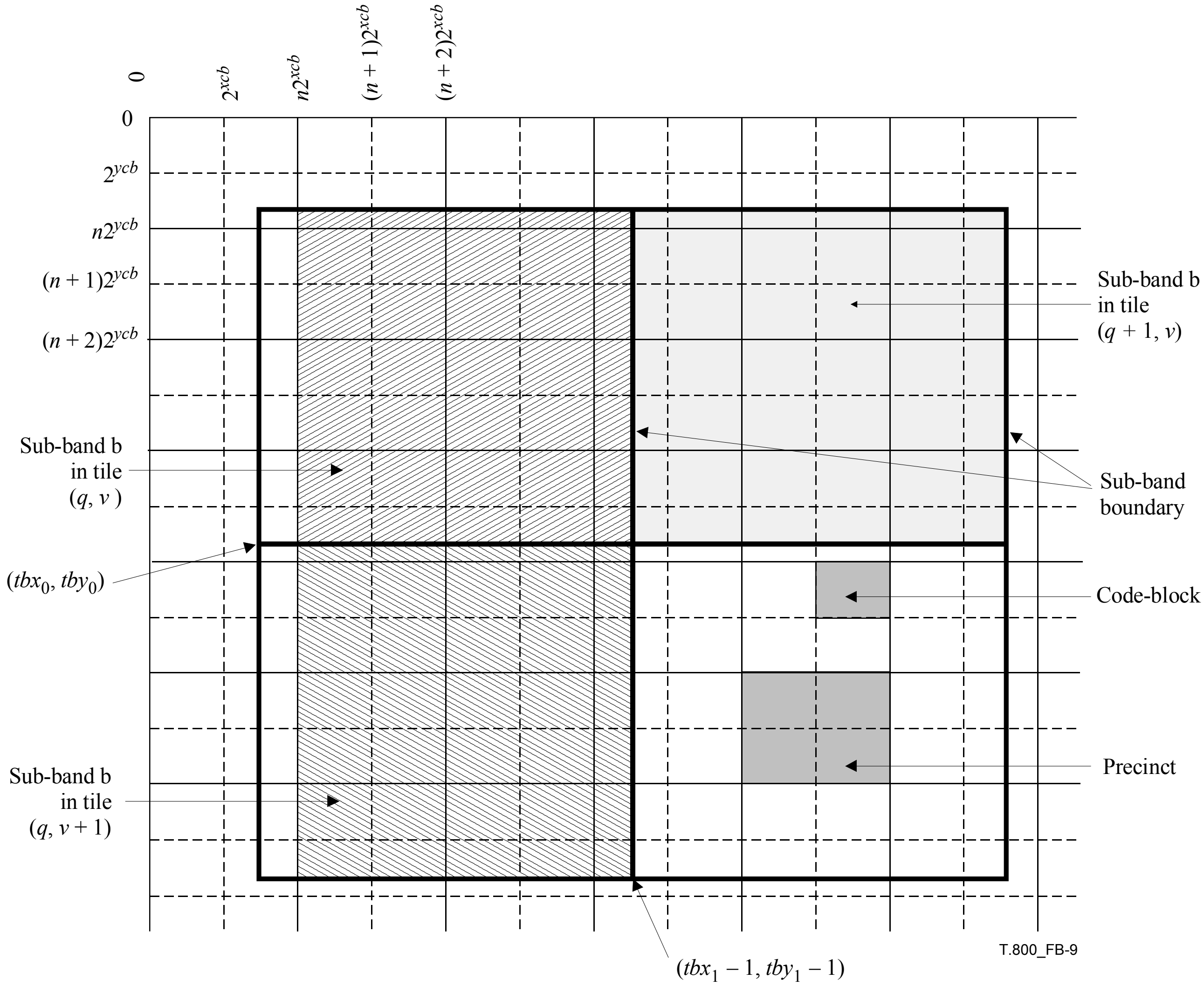


Figure B.9 – Code-blocks and precincts in sub-band *b* from four different tiles

B.8 Layers

The compressed image data of each code-block is distributed across one or more layers in the codestream. Each layer consists of some number of consecutive bit-plane coding passes from each code-block in the tile, including all sub-bands of all components for that tile. The number of coding passes in the layer may vary from code-block to code-block and may be as little as zero for any or all code-blocks. The number of layers for the tile is signalled in the COD marker segment (see A.6.1).

For a given code-block, the first coding pass, if any, in layer n is the coding pass immediately following the last coding pass for the code-block in layer $n – 1$, if any.

NOTE 1 – Each layer successively and monotonically improves the image quality.

Layers are indexed from 0 to $L – 1$, where L is the number of layers in each tile-component.

NOTE 2 – Figure B.10 shows an example of nine precincts of resolution level m. Table B.2 shows the layer formation.

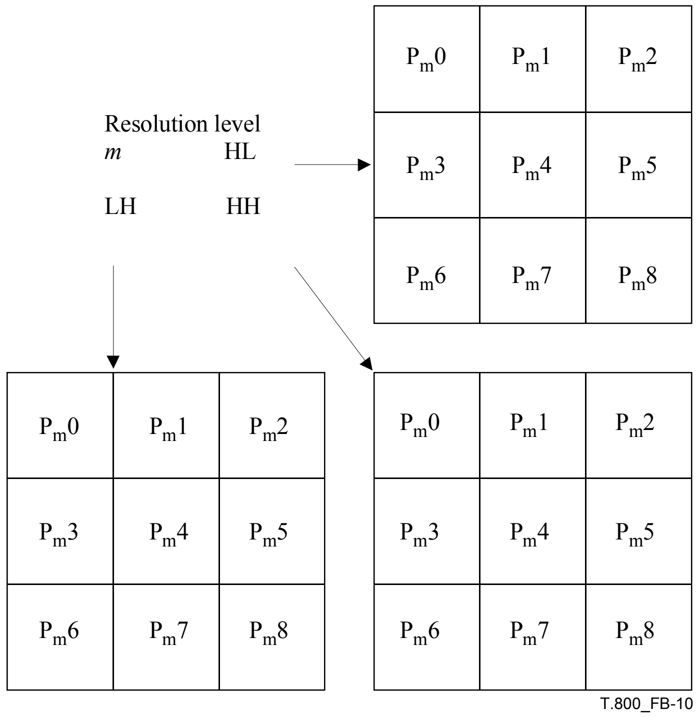


Figure B.10 – Diagram of precincts of one resolution level of one component

Table B.2 – Example of layer formation (only one component shown)

Resolution level	**0**			**...**	**m**				**...**	N_L		
Precinct	P_00	P_01	...	...	P_m0	P_m1	...	P_m8	...	$P_{Nl}0$	$P_{Nl}1$	...
Layer 0	Packet 0	Packet 0	...	...	Packet 0	Packet 0	...	Packet 0	...	Packet 0	Packet 0	...
Layer 1	Packet 1	Packet 1	...	...	Packet 1	Packet 1	...	Packet 1	...	Packet 1	Packet 1	...
...	...		...	...	...	...	...	...	...	...	...	...

The basic building blocks of layers are packets. Packets are created from the code-block compressed image data from the precincts of different resolution levels (for a given tile-component).

B.9 Packets

All compressed image data representing a specific tile, layer, component, resolution level and precinct appears in the codestream in a contiguous segment called a packet. Packet data is aligned at 8-bit (one byte) boundaries.

As defined in F.3.1, resolution level $r = 0$ contains the sub-band coefficients from the N_LLL band, where N_L is the number of decomposition levels. Each subsequent resolution level, $r > 0$, contains the sub-band coefficients from the nHL, nLH, and nHH sub-bands, as defined in Annex F, where $n = N_L - r + 1$. There are $N_L + 1$ resolution levels for a tile-component with N_L decomposition levels.

The compressed image data in a packet is ordered such that the contribution from the LL, HL, LH and HH sub-bands appear in that order. This sub-band order is identical to the order defined in F.3.1. Within each sub-band, the code-block contributions appear in raster order, confined to the bounds established by the relevant precinct. Resolution level $r = 0$ contains only the N_LLL band and resolution levels $r > 0$ contain only the HL, LH and HH bands. Only those code-blocks that contain samples from the relevant sub-band, confined to the precinct, have any representation in the packet.

NOTE 1 – Figure B.11 shows the organization of code-blocks within a precinct that form a packet. Table B.3 shows an example of code-block coding passes that form packets. In Table B.3 the variables a, b, and c are code-block coding passes where a = significance propagation pass, b = magnitude refinement pass, and c = cleanup pass (see Annex D).

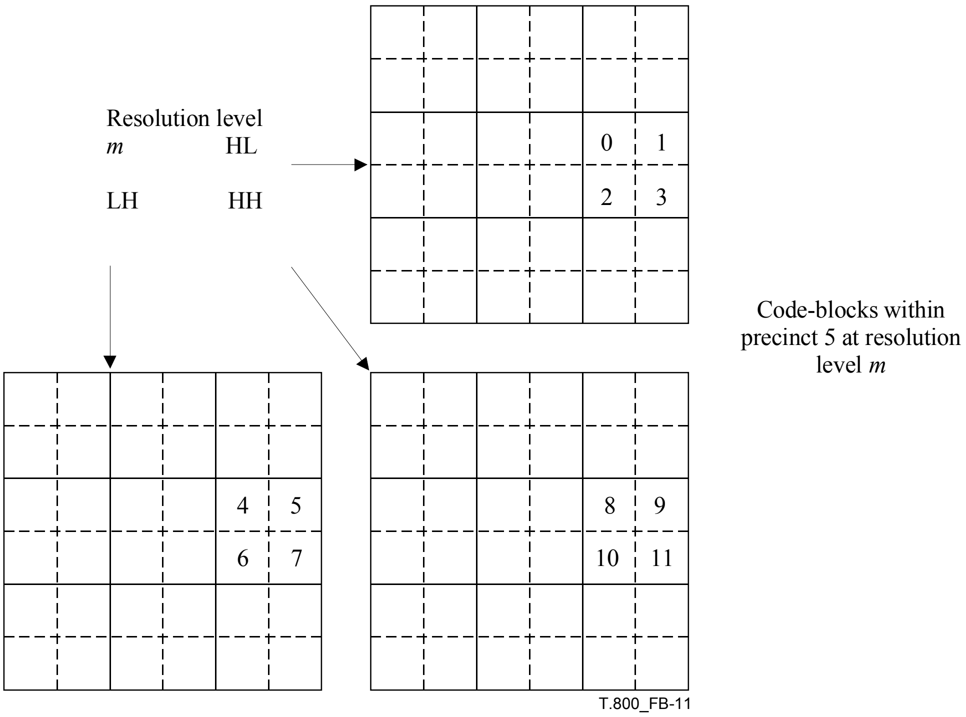


Figure B.11 – Diagram of code-blocks within precincts at one resolution level

Table B.3 – Example of packet formation

	code-block 0	code-block 1	code-block 2	...	code-block 10	code-block 11	
MSB	c	0	0	...	c	0	packet 0
	a	0	0	...	a	0	
	b	0	0	...	b	0	
	c	c	0	...	c	0	
	a	a	0	...	a	0	packet 1
	b	b	0	...	b	0	
	c	c	c	...	c	c	
	...	...	...	...	...	...	etc.
LSB	a	a	a	...	a	a	
	b	b	b	...	b	b	
	c	c	c	...	c	c	

Packet data is introduced by a packet header whose syntax is described in B.10 and is followed by a packet body containing the actual code-bytes contributed by each of the relevant code-blocks. The order defined above is followed in constructing both the packet header and the packet body.

As described in B.6, it can happen that a precinct contains no code-blocks from any of the sub-bands at some resolution level. When this occurs, all packets corresponding to that precinct must appear in the codestream as empty packets, in accordance with the packet header described in B.10.

NOTE 2 – Even when a precinct contains relevant code-blocks, an encoder might choose to include no coding passes whatsoever in the corresponding packet at a given layer. In this case, an empty packet must still appear in the codestream.

B.10 Packet header information coding

The packets have headers with the following information:

- Zero length packet;
- Code-block inclusion;
- Zero bit-plane information;
- Number of coding passes;
- Length of the code-block compressed image data from a given code-block.

Two items in the header are coded with a scheme called tag trees described below. The bits of the packet header are packed into a whole number of bytes with the bit-stuffing routine described in B.10.1.

The packet headers appear in the codestream immediately preceding the packet data, unless one of the PPM or PPT marker segments has been used. If the PPM marker segment is used, all of the packet headers are relocated to the main header (see 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 relocated to tile-part headers (see A.7.5).

B.10.1 Bit-stuffing routine

Bits are packed into bytes from the MSB to the LSB. Once a complete byte is assembled, it is appended to the packet header. If the value of the byte is 0xFF, the next byte includes an extra zero bit stuffed into the MSB. Once all bits of the packet header have been assembled, the last byte is packed to the byte boundary and emitted. The last byte in the packet header shall not be an 0xFF value (thus the single zero bit stuffed after a byte with 0xFF must be included even if the 0xFF would otherwise have been the last byte).

B.10.2 Tag trees

A tag tree is a way of representing a two-dimensional array of non-negative integers in a hierarchical way. It successively creates reduced resolution levels of this two-dimensional array, forming a tree. At every node of this tree the minimum integer of the (up to four) nodes below it is recorded. Figure B.12 shows an example of this representation. The notation, $q_i(m, n)$, is the value at the node that is mth from the left and nth from the top, at the ith level. Level 0 is the lowest level of the tag tree; it contains the top node.

1 $q_3(0, 0)$	3 $q_3(1, 0)$	2 $q_3(2, 0)$	3	2	3
2	2	1	4	3	2
2	2	2	2	1	2

a) Original array of numbers, level 3

1 $q_2(0, 0)$	1 $q_2(1, 0)$	2
2	2	1

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

1 $q_1(0, 0)$	1

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

1 $q_0(0, 0)$

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

T.800_FB-12

Figure B.12 – Example of a tag tree representation

The elements of the array are traversed in raster order for coding. The coding is the answer to a series of questions. Each node has an associated current value, which is initialized to zero (the minimum). A 0 bit in the tag tree means that the minimum (or the value in the case of the highest level) is larger than the current value and a 1 bit means that the minimum (or the value in the case of the highest level) is equal to the current value. For each contiguous 0 bit in the tag tree the current value is incremented by one. Nodes at higher levels cannot be coded until lower level node values are fixed (i.e, a 1 bit is coded). The top node on level 0 (the lowest level) 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 halted when sufficient information has been obtained. Also, the hierarchical nature of the tag trees means that the answers to many questions will have been formed 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 Figure B.12, 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 the 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 $q_3(0, 0)$ and $q_3(1, 0)$ are known, the code bits for $q_3(2, 0)$ will be 101. The first 1 indicates $q_2(1, 0)$ is one. The following 01 then indicates $q_3(2, 0)$ is 2. This process continues for the entire array in Figure B.12a.

B.10.3 Zero length packet

The first bit in the packet header denotes whether the packet has a length of zero (empty packet). The value 0 indicates a zero length; no code-blocks are included in this case. The value 1 indicates a non-zero length; this case is considered exclusively hereinafter.

NOTE – If a packet is marked as empty, then no code-blocks may contribute to the corresponding layer. If the next packet is not marked as empty, the code-block inclusion information (defined in B.10.4) for the previous layer with the empty bit set has to be included. The code-block inclusion information for code-blocks which have not yet been included in any packet is encoded using a tag tree whose entries are initialized with the layer number of the first layer to which the code-block contributes. Thus the tag tree will have redundant information identifying whether or not the code-block contributes to both the current layer and the layer in which the packet was marked as empty.

B.10.4 Code-block inclusion

Information concerning whether or not any compressed image data from each code-block is included in the packet is signalled in one of two different ways depending upon whether or not the same code-block has already been included in a previous packet (i.e., within a previous layer).

For code-blocks that have been included in a previous packet, a single bit is used to represent the information, where a 1 means that the code-block is included in this layer and a 0 means that it is not.

For code-blocks that have not been previously included in any packet, this information is signalled with a separate tag tree code for each precinct as confined to a sub-band. The values in this tag tree are the number of the layer in which the current code-block is first included. Although the exact sequence of bits that represent the inclusion tag tree appears in the bit stream, only the bits needed for determining whether the code-block is included are placed in the packet header. If some of the tag tree is already known from previous code-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.

B.10.5 Zero bit-plane information

If a code-block is included for the first time, the packet header contains information identifying the actual number of bit-planes used to represent coefficients from the code-block. The maximum number of bit-planes available for the representation of coefficients in any sub-band, b, is given by M_b as defined in Equation (E-2). In general, however, the number of actual bit-planes for which coding passes are generated is $M_b - P$, where the number of missing most significant bit-planes, P, may vary from code-block to code-block; these missing bit-planes are all taken to be zero. The value of P is coded in the packet header with a separate tag tree for every precinct, in the same manner as the code-block inclusion information.

B.10.6 Number of coding passes

The number of coding passes included in this packet from each code-block is identified in the packet header using the codewords shown in Table B.4. This table provides for the possibility of signalling up to 164 coding passes.

Table B.4 – 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 to 36	`1111 0000 0`
	to
	`1111 1111 0`

37 to 164	`1111 11111 0000 000`
	to
	`1111 11111 1111 111`

NOTE – Since the value of M_b is limited to a maximum value of 37 by the constraints imposed by the syntax of the QCD and QCC marker segments (see A.6.4, A.6.5, and Equation (E-4)), it is not possible for more than 109 coding passes to be employed by the code-block coding algorithm described in Annex D.

B.10.7 Length of the compressed image data from a given code-block

The packet header identifies the number of bytes contributed by each included code-block. The sequence of bytes actually included for any given code-block must not end in a 0xFF. Thus, in the event that an 0xFF would have appeared at the end of a code-block's contribution to some packet, the 0xFF may be safely moved to the subsequent packet which contains contributions from the code-block, or dropped if there is no such packet. The example coding pass length calculation algorithm described in Annex D ensures that no coding pass will ever be considered as ending with an 0xFF.

NOTE – This is, in fact, not a burdensome requirement, since 0xFFs are always synthesized as necessary by the arithmetic coder described in Annex C.

In signalling the number of bytes contributed by the code-block, there are two cases: the code-block contribution contains a single codeword segment; or the code-block contribution contains multiple codeword segments. Multiple codeword segments arise when a termination occurs between coding passes which are included in the packet, as shown in Tables D.8 and D.9.

B.10.7.1 Single codeword segment

A codeword segment is the number of bytes contributed to a packet by a code-block. The length of a codeword segment is represented by a binary number of length:

$$bits = Lblock + \lfloor \log_2(\text{coding passes added}) \rfloor \qquad \text{(B-19)}$$

where *Lblock* is a code-block state variable. A separate *Lblock* is used for each code-block in the precinct.

The value of *Lblock* is initially set to three. The number of bytes contributed by each code-block is preceded by signalling bits that increase the value of *Lblock*, as needed. A signalling 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 length of the code-block contribution can increase or decrease depending on the number of coding passes included.

NOTE 1 – 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 to *Lblock*, $\log_2 2 = 1$, 101100 = 44), and 1 0 10000110 (10 adds one bit to Lblock, $\log_2 5 = 2$, 10000110 = 134).

NOTE 2 – There is no requirement that the minimum number of bits be used to signal length (any number is valid).

B.10.7.2 Multiple codeword segments

Let T be the set of indices of terminated coding passes included for the code-block in the packet as indicated in Tables D.8 and D.9. If the index final coding pass included in the packet is not a member of T, then it is added to T. Let $n_1 < ... < n_K$ be the indices in T. K lengths are signalled consecutively with each length using the mechanism described in B.10.7.1. The first length is the number of bytes from the start of the code-block's contribution in this packet to the end of coding pass n_1. The number of added coding passes for the purposes of Equation (B-19) is the number of passes in the packet up through 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. The number of added coding passes for the purposes of Equation (B-19) is $n_2 - n_1$. This procedure is repeated for all *K* lengths.

NOTE – Consider the selective arithmetic coding bypass (see D.6). Say that the passes included in a packet for a given code-block are the cleanup pass of bit-plane number 4 through the significance propagation pass of bit-plane number 6 (see Table D.9). These passes are indexed as {0, 1, 2, 3, 4} and the lengths are given as {6, 31, 44, 134, 192} respectively. Then T = {0, 2, 3, 4} and K = 4 lengths are signalled. The set of lengths to be signalled is {6, 75, 134, 192} and the corresponding number of coding passes that are added is {1, 2, 1, 1}. A valid code bit sequence is 11 1110 (*Lblock* increased to 8), 0000 0110 ($\log_2 1 = 0$, 8 bits used to code length of 6), 0 0100 1011 ($\log_2 2 = 1$, 9 bits used to code the length of 75), 1000 0110 ($\log_2 1 = 0$, 8 bits used to code the length of 134), and 1100 0000 ($\log_2 1 = 0$, 8 bits used to code the length of 192). Notice that the value of *Lblock* is incremented only at the start of the sequence.

B.10.8 Order of information within packet header

The following is the packet header information order for one packet of a specific layer, tile-component, resolution level and precinct.

bit for zero or non-zero length packet
for each sub-band (LL or HL, LH and HH)
    for all code-blocks in this sub-band confined to the relevant precinct, 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 (*Lblock*)
            for each codeword segment
            length of codeword segment

The packet header may be immediately followed by the EPH marker as described in A.8.2. The EPH marker may appear regardless of whether the packet contains any code-block contributions. In the event that the packet header appears in a PPM or PPT marker segment, the EPH marker (if used) must appear together with the packet header.

NOTE – Figure B.13 and Table B.5 show a brief example of packet header construction. Figure B.13 shows the information known to the encoder. In particular the "inclusion information" shows the layer where each code-block first appears in a packet. The decoder will receive this information via the inclusion tag tree in several packet headers. Table B.5 shows the resulting bit stream (in part) from this information.

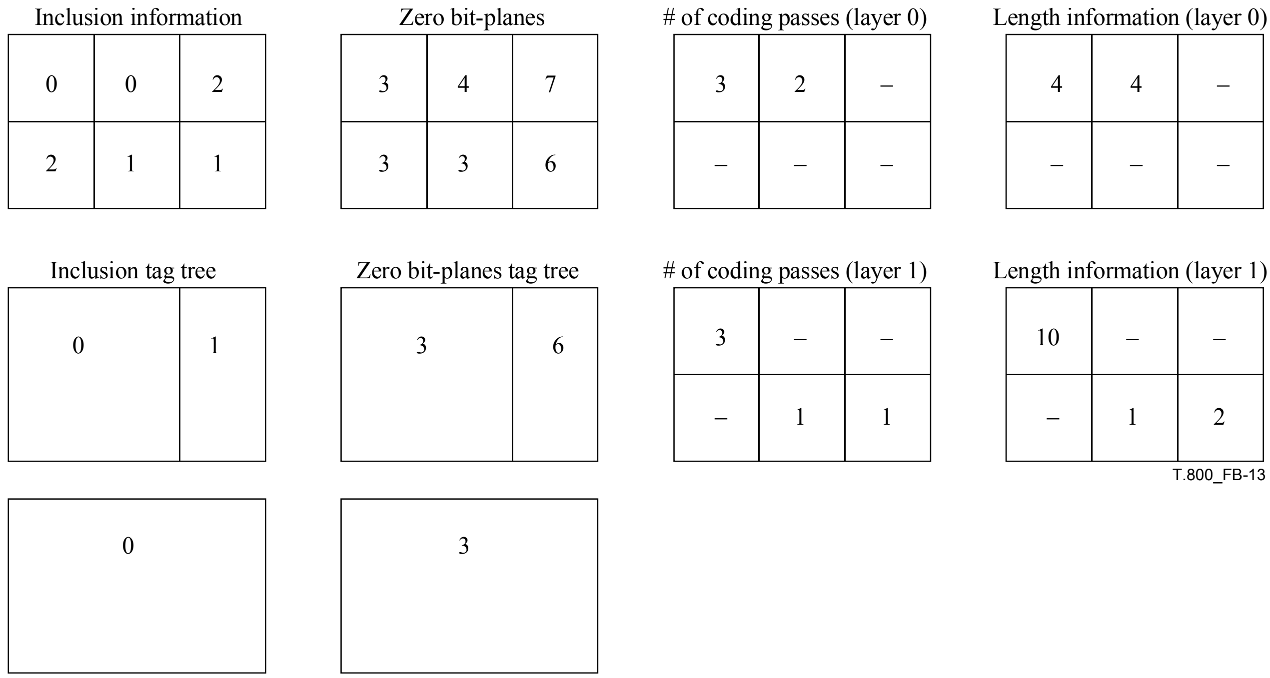


Figure B.13 – Example of the information known to the encoder

Table B.5 – Example packet header bit stream

Bit stream (in order)	Derived meaning
1	Packet non-zero in length
111	Code-block 0, 0 included for the first time (partial inclusion tag tree)
000111	Code-block 0, 0 insignificant for 3 bit-planes
1100	Code-block 0, 0 has 3 coding passes included
0	Code-block 0, 0 length indicator is unchanged
0100	Code-block 0, 0 has 4 bytes, 4 bits are used, 3 + floor($\log_2 3$)
1	Code-block 1, 0 included for the first time (partial inclusion tag tree)
01	Code-block 1, 0 insignificant for 4 bit-planes
10	Code-block 1, 0 has 2 coding passes included
10	Code-block 1, 0 length indicator is increased by 1 bit (3 to 4)
00100	Code-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	Code-block 2, 0 not yet included (partial tag tree)
0	Code-block 0, 1 not yet included
0	Code-block 1, 1 not yet included
	Code-block 2, 1 not yet included (no data needed, already conveyed by partial tag tree for code-block 2, 0)
•••	Packet header data for the other sub-bands, packet data

Packet for the next layer

Bit stream (in order)	Derived meaning
1	Packet non-zero in length
1	Code-block 0, 0 included again
1100	Code-block 0, 0 has 3 coding passes included
0	Code-block 0, 0 length indicator is unchanged
1010	Code-block 0, 0 has 10 bytes, 3 + $\log_2$ (3) bits used
0	Code-block 1, 0 not included in this layer
10	Code-block 2, 0 not yet included
0	Code-block 0, 1 not yet included
1	Code-block 1, 1 included for the first time
1	Code-block 1, 1 insignificant for 3 bit-planes
0	Code-block 1, 1 has 1 coding passes included
0	Code-block 1, 1 length information is unchanged
001	Code-block 1, 1 has 1 byte, 3 + $\log_2$ (1) bits used
1	Code-block 2, 1 included for the first time
00011	Code-block 2, 1 insignificant for 6 bit-planes
0	Code-block 2, 1 has 1 coding passes included
0	Code-block 2, 1 length indicator is unchanged
010	Code-block 2, 1 has 2 bytes, 3 + $\log_2$ 1 bits used
•••	Packet header data for the other sub-bands, packet data

B.11 Tile and tile-parts

Each coded tile is represented by a sequence of packets. The rules governing the order that the packets of a tile appear within the codestream is specified in B.12. It is possible for a tile to contain no packets, in the event that no samples from any image component map to the region occupied by the tile on the reference grid.

Any tile's representation may be truncated by discarding one or more trailing bytes. Also, any number of whole packets (in order) may be dropped and the final packet appearing in the tile may be partially truncated. The tile length marker segment parameters shall reflect this.

The sequence of packets representing any particular tile may be divided into contiguous segments known as tile-parts. Any number of packets (including zero) may be contained in a tile-part. Each tile must contain at least one tile-part. The divisions between tile-parts must occur at packet boundaries. While tiles are coherent geometric areas, the tile-parts may be distributed throughout the codestream in any desired fashion, provided tile-parts from the same tile appear in the order that preserves the original packet sequence. Each tile-part commences with an SOT marker segment (see A.4.2), containing the index of the tile to which the tile-part belongs.

NOTE – It is possible to interleave tile-parts from different tiles, as long as the order of the tile-parts from every tile is preserved. For example, a legitimate codestream might have the following order:

– Tile number 0, tile-part number 0;
– Tile number 1, tile-part number 0;
– Tile number 0, tile-part number 1;
– Tile number 1, tile-part number 1;
– etc.

If SOP marker segments are allowed (by signalling in the COD marker segment, see A.6.1), each packet in any given tile-part may be appended with an SOP marker segment (see A.8.1). However, whether or not the SOP marker segment is used, the count in the Nsop is incremented for each packet. If the packet headers are moved to a PPM or PPT marker segments (see A.7.4 and A.7.5), then the SOP marker segments may appear immediately before the packet body in the tile-part compressed image data portion.

If EPH markers are required (by signalling in the COD marker segment, see A.6.1), each packet header in any given tile-part shall postpended with an EPH marker segment (see A.8.2). If the packet headers are moved to a PPM or PPT marker segments (see A.7.4 and A.7.5), then the EPH markers shall appear after the packet headers in the PPM or PPT marker segments.

B.12 Progression order

For a given tile-part, the packets contain all compressed image data from a specific layer, a specific component, a specific resolution level, and a specific precinct. The order in which these packets are found in the codestream is called the progression order. The ordering of the packets can progress along four axes: layer, component, resolution level and precinct.

It is possible that components have a different number of resolution levels. In this case, the resolution level that corresponds to the N_LLL sub-band is the first resolution level ($r = 0$) for all components. The indices are synchronized from that point on.

NOTE – For example, take the case of resolution level-position-component-layer progression and two components with 7 resolution levels (6 decomposition levels) and 3 resolution levels (2 decomposition levels) respectively. The $r = 0$ will correspond to the N_LLL sub-band of both components. From $r = 0$ to $r = 2$ the components will be interleaved as described below. From $r = 3$ to $r = 6$ only component 0 will have packets.

B.12.1 Progression order determination

The COD marker segments signal which of the five progression orders are used (see A.6.1). The progression order can also be overridden with the POC marker segment (see A.6.6) in any tile-part header. For each of the possible progression orders the mechanism to determine the order in which packets are included is described below.

B.12.1.1 Layer-resolution level-component-position progression

Layer-resolution level-component-position progression is defined as the interleaving of the packets in the following order:

for each $l = 0,..., L - 1$

for each $r = 0,..., N_{max}$

for each $i = 0,...,$ Csiz – 1

for each $k = 0,..., numprecincts - 1$

packet for component i, resolution level r, layer l, and precinct k.

Here, L is the number of layers and N_{max} is the maximum number of decomposition levels, N_L, used in any component of the tile. A progression of this type might be useful when low sample accuracy is most desirable, but information is needed for all components.

B.12.1.2 Resolution level-layer-component-position progression

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

for each $r = 0,..., N_{max}$

for each $l = 0,..., L - 1$

for each $i = 0,...,$ Csiz – 1

for each $k = 0,..., numprecincts - 1$

packet for component i, resolution level r, layer l, and precinct k.

A progression of this type might be useful in providing low resolution level versions of all image components.

B.12.1.3 Resolution level-position-component-layer progression

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

for each $r = 0,..., N_{max}$

for each $y = ty_0,..., ty_1 - 1$,

for each $x = tx_0,..., tx_1 - 1$,

for each $i = 0,...,$ Csiz – 1

if ((y divisible by $YRsiz(i) \cdot 2^{PPy(r,i)+N_L(i)-r}$) OR (($y = ty_0$) AND ($try_0 \cdot 2^{N_L(i)-r}$ NOT divisible by $2^{PPy(r,i)+N_L(i)-r}$))

if ((x divisible by $XRsiz(i) \cdot 2^{PPx(r,i)+N_L(i)-r}$) OR (($x = tx_0$) AND ($trx_0 \cdot 2^{N_L(i)-r}$ NOT divisible by $2^{PPx(r,i)+N_L(i)-r}$))

for the next precinct, k, if one exists,

for each $l = 0,..., L - 1$

packet for component i, resolution level r, layer l, and precinct k.

In the above, k can be obtained from:

$$k = \left\lfloor \frac{\left\lceil \dfrac{x}{XRsiz(i) \cdot 2^{N_L - r}} \right\rceil}{2^{PP_x(r,i)}} \right\rfloor - \left\lfloor \frac{trx_0}{2^{PP_x(r,i)}} \right\rfloor + numprecinctswide(r,i) \cdot \left(\left\lfloor \frac{\left\lceil \dfrac{y}{YRsiz(i) \cdot 2^{N_L - r}} \right\rceil}{2^{PP_y(r,i)}} \right\rfloor - \left\lfloor \frac{try_0}{2^{PP_y(r,i)}} \right\rfloor \right) \quad \text{(B-20)}$$

To use this progression, XRsiz and YRsiz values must be powers of two for each component. A progression of this type might be useful in providing low resolution level versions of all image components at a particular spatial location.

NOTE – The iteration of variables x and y in the above formulation is given for simplicity only of expression, not implementation. Most of the (x, y) pairs generated by this loop will generally result in the inclusion of no packets. More efficient iterations can be found based upon the minimum of the dimensions of the various precincts, mapped into the reference grid. This note also applies to the loops given for the following two progressions.

B.12.1.4 Position-component-resolution level-layer progression

Position-component-resolution level-layer 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 $r = 0,..., N_L$ where N_L is the number of decomposition levels for component i,

if ((y divisible by $YRsiz(i) \cdot 2^{PPy(r,i)+N_L(i)-r}$) OR (($y = ty_0$) AND ($try_0 \cdot 2^{N_L(i)-r}$ NOT divisible by $2^{PPy(r,i)+N_L(i)-r}$))

if ((x divisible by $XRsiz(i) \cdot 2^{PPx(r,i)+N_L(i)-r}$) OR (($x = tx_0$) AND ($trx_0 \cdot 2^{N_L(i)-r}$ NOT divisible by $2^{PPx(r,i)+N_L(i)-r}$))

for the next precinct, k, if one exists, in the sequence shown in Figure B.8

for each $l = 0,...,$ L – 1

packet for component i, resolution level r, layer l, and precinct k.

In the above, k can be obtained from Equation (B-20). To use this progression, XRsiz and YRsiz values shall be powers of two for each component. A progression of this type might be useful in providing high sample accuracy for a particular spatial location in all components.

B.12.1.5 Component-position-resolution level-layer progression

Component-position-resolution level-layer 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_L where N_L is the number of decomposition levels for component i,

if ((y divisible by $YRsiz(i) \cdot 2^{PPy(r,i)+N_L(i)-r}$) OR (($y = ty_0$) AND ($try_0 \cdot 2^{N_L(i)-r}$ NOT divisible by $2^{PPy(r,i)+N_L(i)-r}$))

if ((x divisible by $XRsiz(i) \cdot 2^{PPx(r,i)+N_L(i)-r}$) OR (($x = tx_0$) AND ($trx_0 \cdot 2^{N_L(i)-r}$ NOT divisible by $2^{PPx(r,i)+N_L(i)-r}$))

for the next precinct, k, if one exists, in the sequence shown in Figure B.8

for each $l = 0,...,$ L – 1

packet for component i, resolution level r, layer l, and precinct k.

In the above, k can be obtained from Equation (B-20). A progression of this type might be useful in providing high accuracy for a particular spatial location in a particular image component.

B.12.2 Progression order volumes

The progression order default is signalled in the COD marker segment in the main header or tile headers (see A.6.1). The progression loops of B.12.1 all go from zero to the maximum value.

If this progression order is to be changed, the POC marker segment is used (see A.6.6). In this case, the "for loops" described in B.12.1 are limited by start points (CSpoc, RSpoc, Layer = 0, inclusive) and end points (CEpoc, REpoc and LEpoc, exclusive). This creates a progression order volumne of packets. All the packets included in the entire progression order volume are found in order in the codestream before the next progression order change takes effect. No packet is ever repeated in the codestream. Therefore, the layer always starts with the next one for a given tile-component, resolution level, and precinct. The decoder is required to determine the next layer.

Thus, the variables in the above loops are bounded by the progression order volumne as described in Equation (B-21).

$$\begin{aligned} CSpod &\le i < CEpod \\ RSpod &\le r < REpod \\ 0 &\le l < LEpod \end{aligned} \qquad \text{(B-21)}$$

NOTE – Figure B.14 shows an example of two progression volumes for a single component image. First packets are sent in resolution level-layer-component-position progression until the box labeled "First" in the figure is complete; then packets are sent in layer-resolution level-component-position progression for the layers of all resolution levels which were not previously sent.

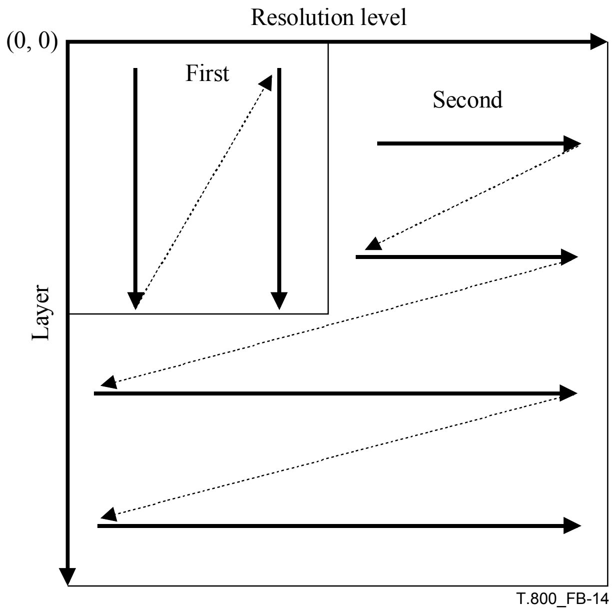


Figure B.14 – Example of progression order volume in two dimensions

B.12.3 Progression order change signalling

If there is a progression order change, then at least one POC marker segment shall be used in the codestream (see A.6.6). There can only be one POC marker segment in a given header (main or tile-part) but that marker segment can describe many progression order changes.

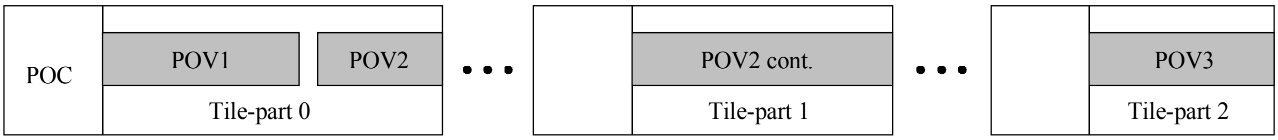
If the POC marker segment is found in the main header, it overrides the progression found in the COD for all tiles. The main header POC marker segement is used for tiles that do not have POC marker segments in their tile-part headers.

If a POC marker segment is used for an individual tile, there shall be a POC marker in the first tile-part header of that tile and all of the progression order changes shall be signalled in the tile-part headers of that tile. The COD progression order and the main header POC marker segment (if there is one) are overridden.

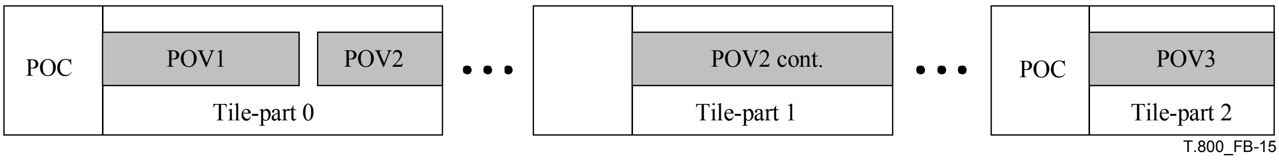
If there are progression order changes signalled by POC marker segments (whether in the main header or the tile-part headers), then all the order of all the packets in the codestream, or the affected tile-parts of the codestream shall be described by progression order volumes in the POC marker segments. There will never be the case where a progression order volume is filled and the next one is not defined. On the other hand, the POC marker segments may describe more progression order volumes than exist in the codestream. Also, the last progression order volume in each tile may be incomplete.

The POC marker segments shall describe progression order volumes in order in any tile-part header before the first included packet appears. However, the POC marker may be, but is not required to be, in the tile-part header immediately before the progression order volume is used. It is possible to describe many progression order volumes in a tile-part header even though those progression order volumes do not appear until later tile-parts.

NOTE – For example, all of the progression order volumes can be described one POC marker segement in the first tile-part header of a tile. Figure B.15a shows this scenario. Equally acceptable, in this case, is describing two progression order volumes in the first tile-part header and one in the third, as shown in Figure B.15b.



a) All of the progression order volumes are described in the POC marker segments in the first tile-part header



b) Progression order volumes 1 and 2 are described in the POC marker segments in the first tile-part header, progression order volume 3 described in the third tile-part header

Figure B.15 – Example of the placement of POC marker segments

Annex C

Arithmetic entropy coding

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

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

C.1 Binary encoding (informative)

Figure C.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 image 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 Recommendation International Standard, CX is a label for a context.

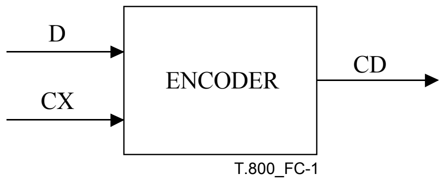


Figure C.1 – Arithmetic encoder inputs and outputs

C.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 image 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.

C.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 compressed image data is removed from the high order bits of the C-register and placed in an external compressed image data buffer. Carry-over into the external buffer is prevented 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:

$$A - (Qe * A) = \text{sub-interval for the MPS} \tag{C-1}$$

$$Qe * A = \text{sub-interval for the LPS} \tag{C-2}$$

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

$$A - Qe = \text{sub-interval for the MPS} \tag{C-3}$$

$$Qe = \text{sub-interval for the LPS} \tag{C-4}$$

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 or MPS provide an approximate symbol counting mechanism which is used to directly estimate the probabilities.

C.2 Description of the arithmetic encoder (informative)

The ENCODER (Figure C.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 image data are output when necessary. When all of the CX and D pairs have been read, 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.88 | ISO/IEC 14492, it is informative in this Recommendation | International Standard. Other methods, such as that defined in D.4.2, are acceptable.

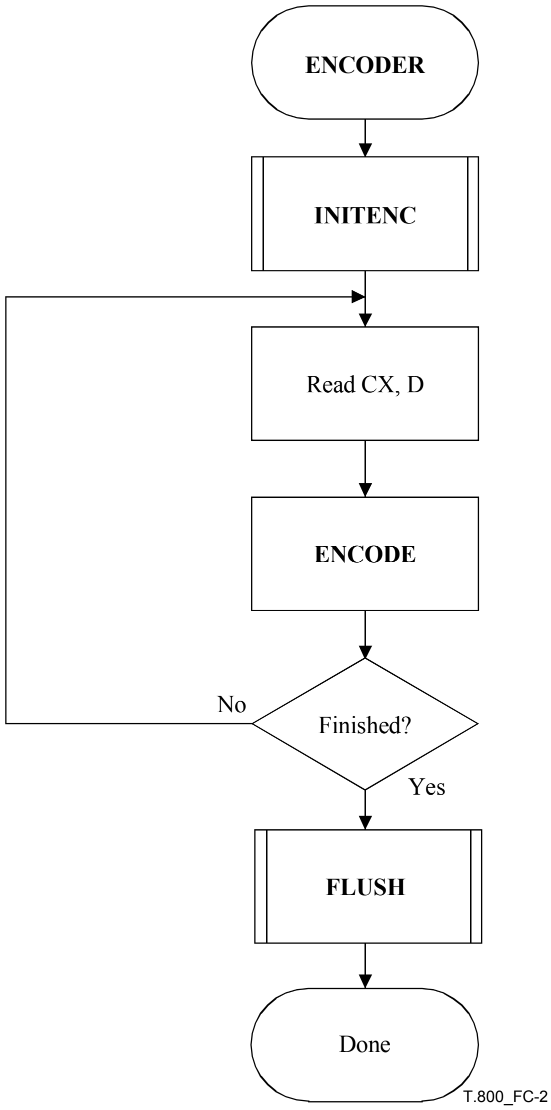


Figure C.2 – Encoder for the MQ-coder

C.2.1 Encoder code register conventions (informative)

The flow charts given in this annex assume the register structures for the encoder shown in Table C.1.

Table C.1 – Encoder register structures

	MSB			LSB
C-register	0000 cbbb	bbbb bsss	xxxx xxxx	xxxx xxxx
A-register	0000 0000	0000 0000	1aaa 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 compressed image 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.

C.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 C.3 shows this procedure.

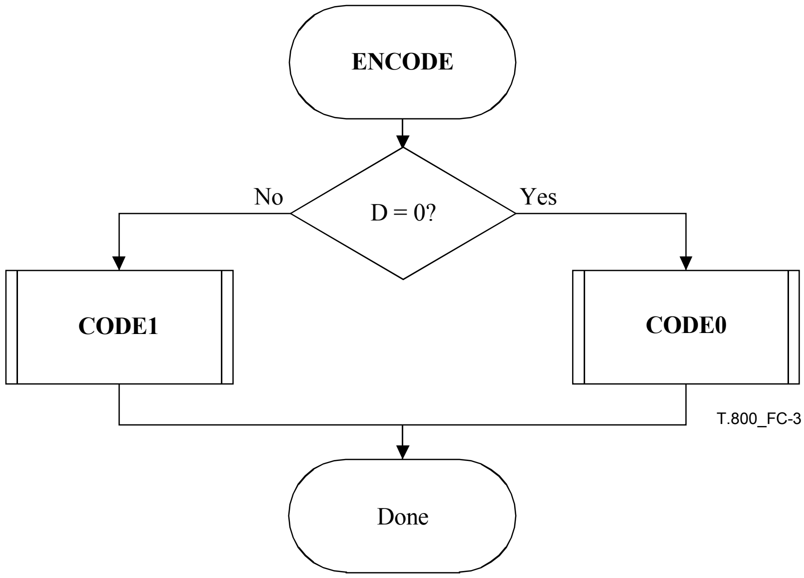


Figure C.3 – ENCODE procedure

C.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 Figures C.4 and C.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.

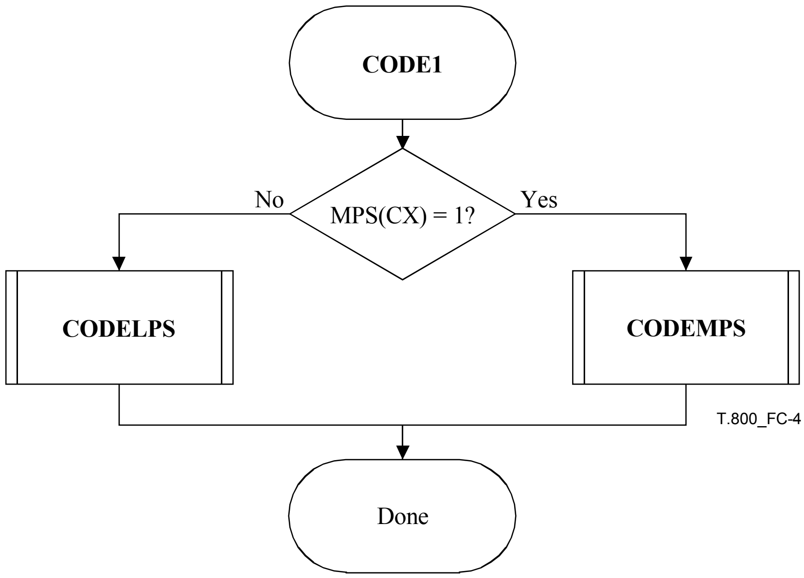


Figure C.4 – CODE1 procedure

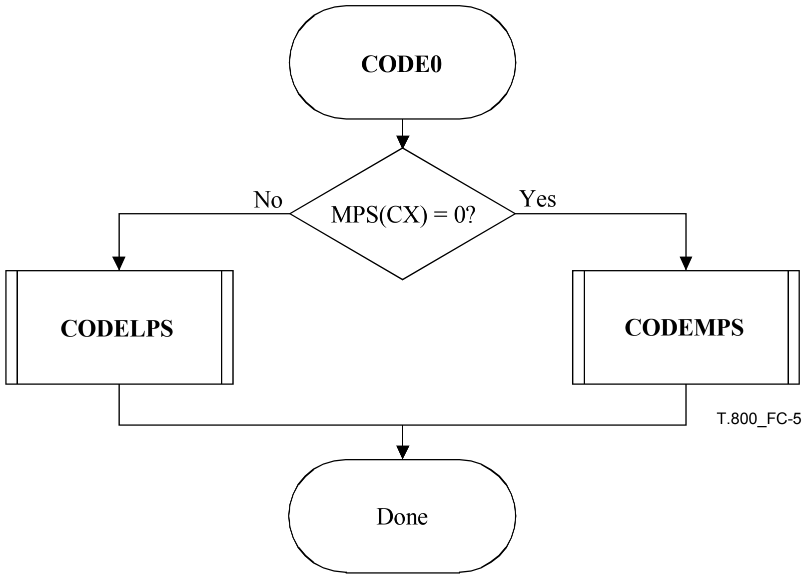


Figure C.5 – CODE0 procedure

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

The CODELPS (Figure C.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 C.2.

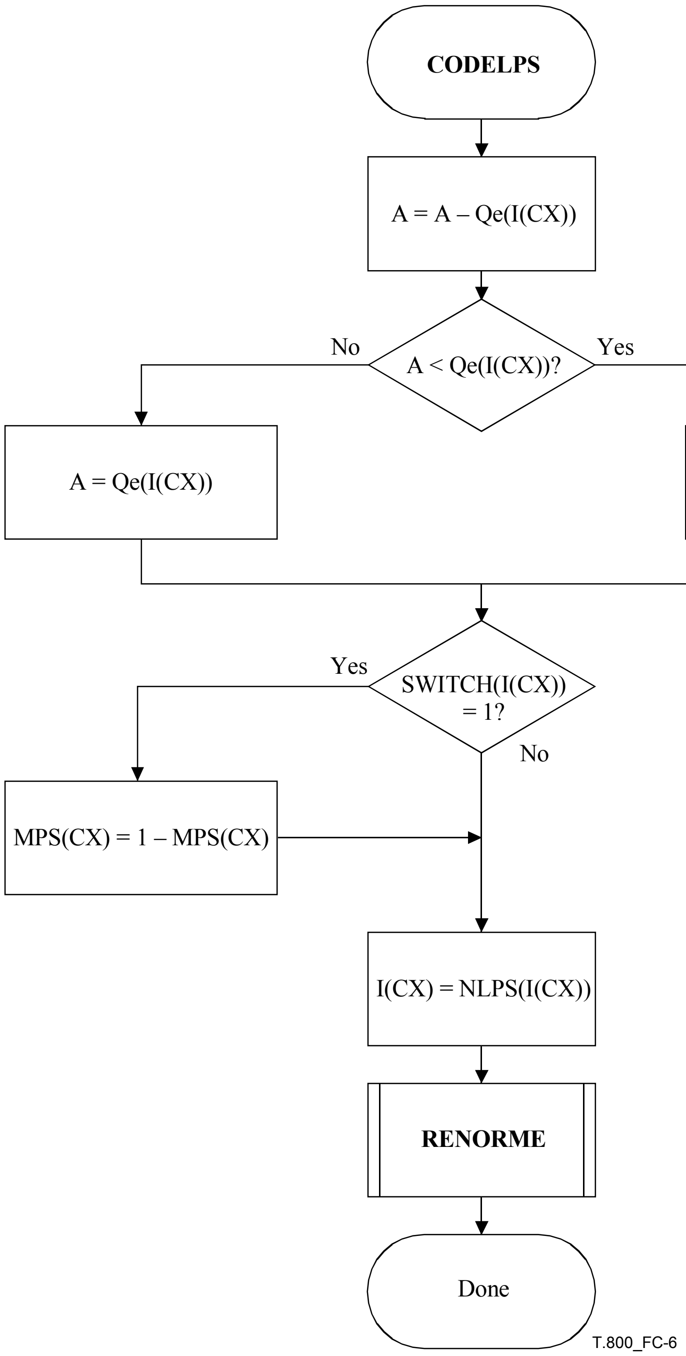


Figure C.6 – CODELPS procedure with conditional MPS/LPS exchange

Table C.2 – Qe values and probability estimation

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
0	0x5601	0101 0110 0000 0001	0,503 937	1	1	1
1	0x3401	0011 0100 0000 0001	0,304 715	2	6	0
2	0x1801	0001 1000 0000 0001	0,140 650	3	9	0
3	0x0AC1	0000 1010 1100 0001	0,063 012	4	12	0
4	0x0521	0000 0101 0010 0001	0,030 053	5	29	0
5	0x0221	0000 0010 0010 0001	0,012 474	38	33	0
6	0x5601	0101 0110 0000 0001	0,503 937	7	6	1

Table C.2 – Qe values and probability estimation *(concluded)*

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
7	0x5401	0101 0100 0000 0001	0,492 218	8	14	0
8	0x4801	0100 1000 0000 0001	0,421 904	9	14	0
9	0x3801	0011 1000 0000 0001	0,328 153	10	14	0
10	0x3001	0011 0000 0000 0001	0,281 277	11	17	0
11	0x2401	0010 0100 0000 0001	0,210 964	12	18	0
12	0x1C01	0001 1100 0000 0001	0,164 088	13	20	0
13	0x1601	0001 0110 0000 0001	0,128 931	29	21	0
14	0x5601	0101 0110 0000 0001	0,503 937	15	14	1
15	0x5401	0101 0100 0000 0001	0,492 218	16	14	0
16	0x5101	0101 0001 0000 0001	0,474 640	17	15	0
17	0x4801	0100 1000 0000 0001	0,421 904	18	16	0
18	0x3801	0011 1000 0000 0001	0,328 153	19	17	0
19	0x3401	0011 0100 0000 0001	0,304 715	20	18	0
20	0x3001	0011 0000 0000 0001	0,281 277	21	19	0
21	0x2801	0010 1000 0000 0001	0,234 401	22	19	0
22	0x2401	0010 0100 0000 0001	0,210 964	23	20	0
23	0x2201	0010 0010 0000 0001	0,199 245	24	21	0
24	0x1C01	0001 1100 0000 0001	0,164 088	25	22	0
25	0x1801	0001 1000 0000 0001	0,140 650	26	23	0
26	0x1601	0001 0110 0000 0001	0,128 931	27	24	0
27	0x1401	0001 0100 0000 0001	0,117 212	28	25	0
28	0x1201	0001 0010 0000 0001	0,105 493	29	26	0
29	0x1101	0001 0001 0000 0001	0,099 634	30	27	0
30	0x0AC1	0000 1010 1100 0001	0,063 012	31	28	0
31	0x09C1	0000 1001 1100 0001	0,057 153	32	29	0
32	0x08A1	0000 1000 1010 0001	0,050 561	33	30	0
33	0x0521	0000 0101 0010 0001	0,030 053	34	31	0
34	0x0441	0000 0100 0100 0001	0,024 926	35	32	0
35	0x02Al	0000 0010 1010 0001	0,015 404	36	33	0
36	0x0221	0000 0010 0010 0001	0,012 474	37	34	0
37	0x0141	0000 0001 0100 0001	0,007 347	38	35	0
38	0x0111	0000 0001 0001 0001	0,006 249	39	36	0
39	0x0085	0000 0000 1000 0101	0,003 044	40	37	0
40	0x0049	0000 0000 0100 1001	0,001 671	41	38	0
41	0x0025	0000 0000 0010 0101	0,000 847	42	39	0
42	0x0015	0000 0000 0001 0101	0,000 481	43	40	0
43	0x0009	0000 0000 0000 1001	0,000 206	44	41	0
44	0x0005	0000 0000 0000 0101	0,000 114	45	42	0
45	0x0001	0000 0000 0000 0001	0,000 023	45	43	0
46	0x5601	0101 0110 0000 0001	0,503 937	46	46	0

C.2.5 Probability estimation

Table C.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 C.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. If the switch is 1, the MPS symbol sense is reversed.

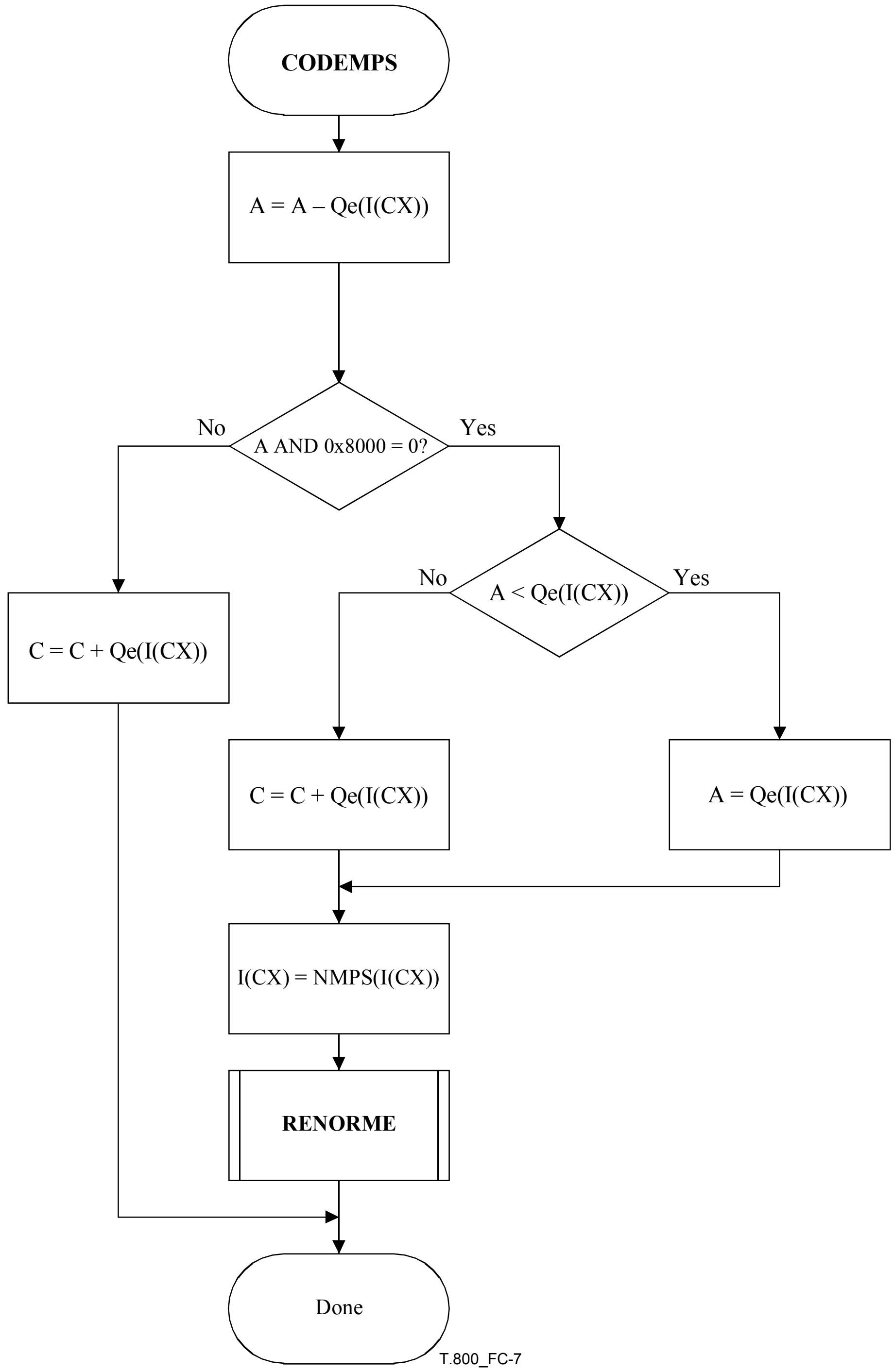


Figure C.7 – CODEMPS procedure with conditional MPS/LPS exchange

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.

C.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 C.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 image data is removed from C by the procedure BYTEOUT. Renormalization continues until A is no longer less than 0x8000.

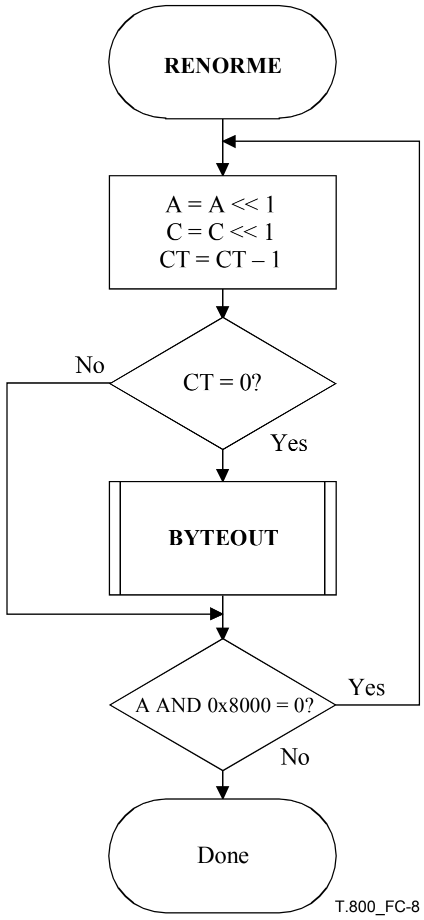


Figure C.8 – Encoder renormalization procedure

C.2.7 Compressed image data output (BYTEOUT) (informative)

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

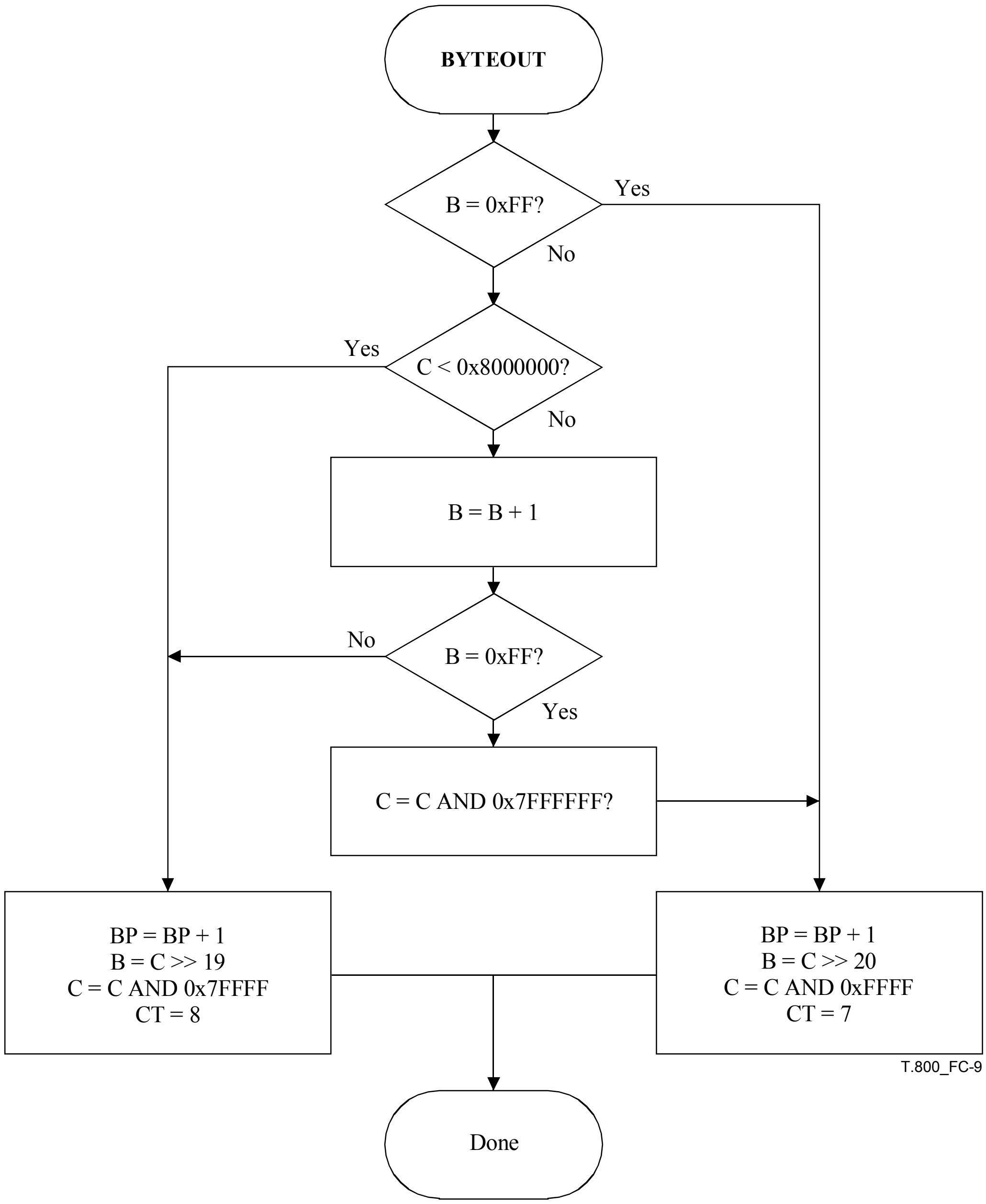


Figure C.9 – BYTEOUT procedure for encoder

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 image 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.

C.2.8 Initialization of the encoder (INITENC) (informative)

The INITENC procedure is used to start the arithmetic coder. After MPS and I are initialized, the basic steps are shown in Figure C.10.

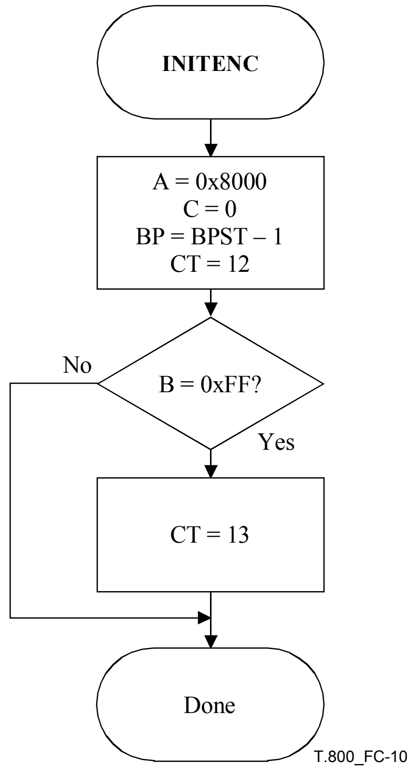


Figure C.10 – Initialization of the encoder

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. 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 initial settings for MPS and I are shown in Table D.7.

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

The FLUSH procedure shown in Figure C.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 image data. This guarantees that any marker code at the end of the compressed image data will be recognized and interpreted before decoding is complete.

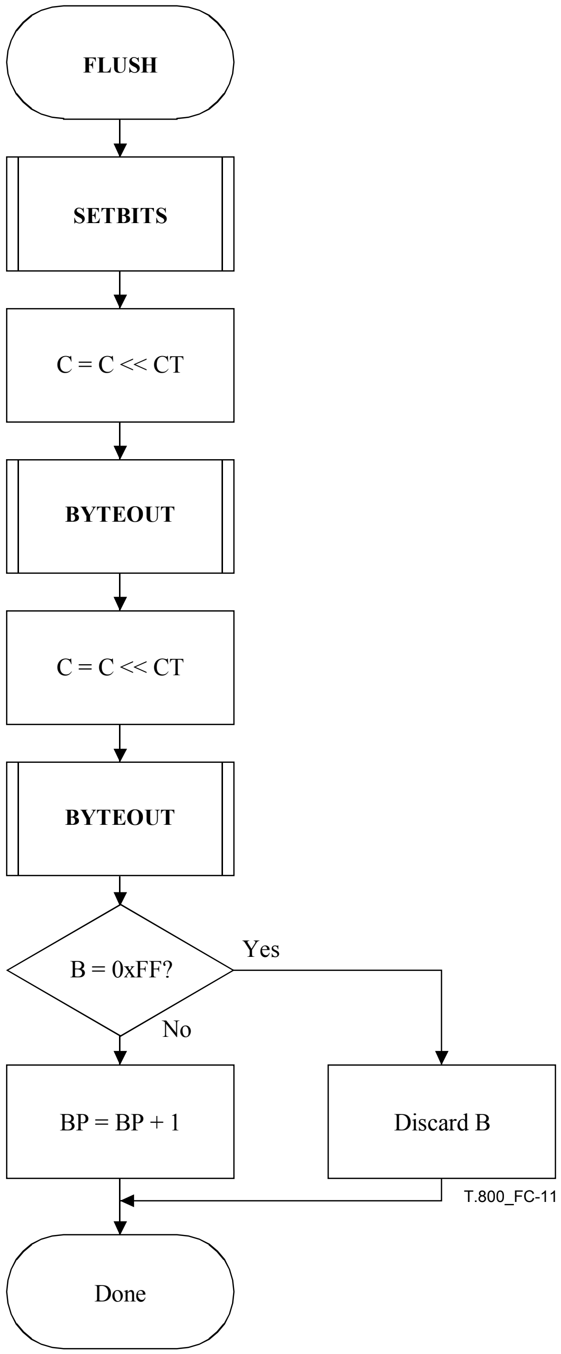


Figure C.11 – FLUSH procedure

The first part of the FLUSH procedure sets as many bits in the C-register to 1 as possible as shown in Figure C.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.

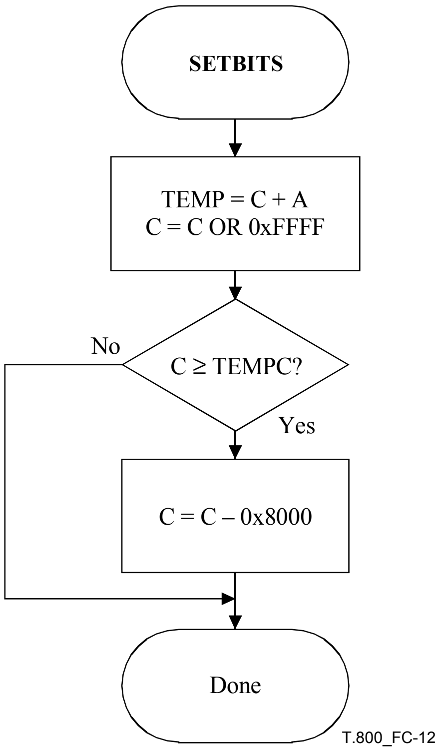


Figure C.12 – Setting the final bits in the C register

The byte in the C-register is then completed by shifting C, and two bytes are then removed. If the byte in buffer, B, is 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.88 | ISO/IEC 14492. However, further reduction of the bit stream is allowed in this Recommendation | International Standard provided correct decoding is assured (see D.4.2).

C.3 Arithmetic decoding procedure

Figure C.13 shows a simple block diagram of a binary adaptive arithmetic decoder. The compressed image 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.

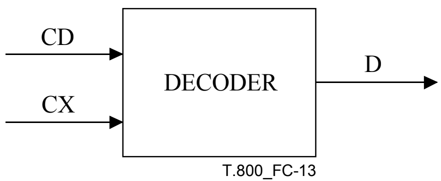


Figure C.13 – Arithmetic decoder inputs and outputs

The DECODER (Figure C.14) initializes the decoder through INITDEC. Contexts, CX, and bytes of compressed image 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, the compressed image data has been decompressed.

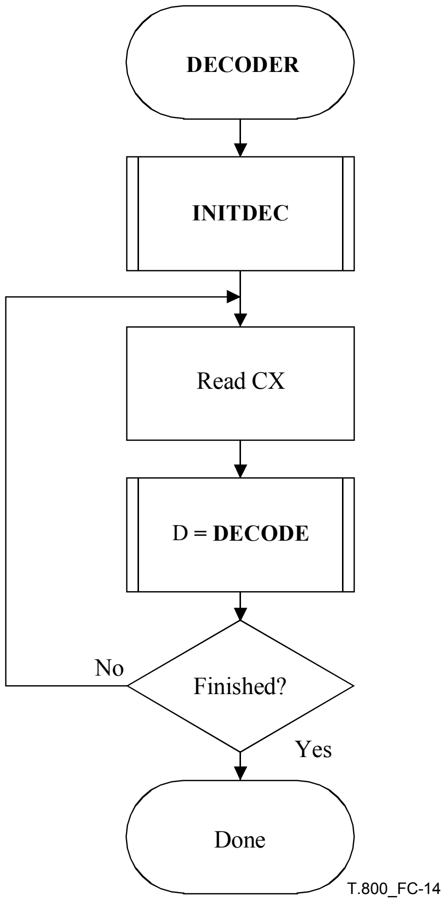


Figure C.14 – Decoder for the MQ-coder

C.3.1 Decoder code register conventions

The flow charts given in this annex assume the register structures for the decoder shown in Table C.3.

Table C.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 clause assume precisions greater than 16 bits. Logical comparisons can be used with 16-bit precision.

C.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 image data that the encoder added. The amount left in the compressed image 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 C.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.

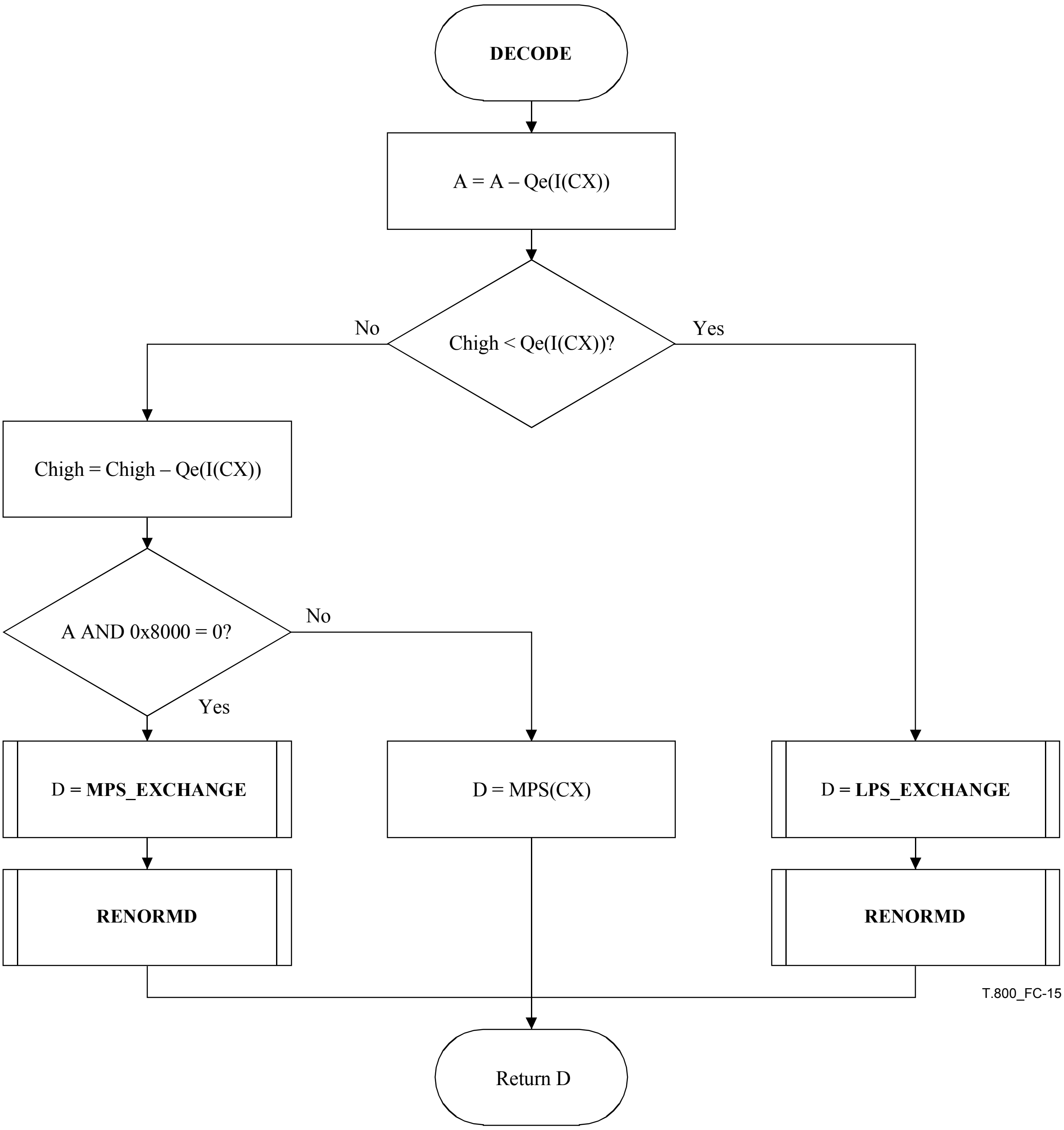


Figure C.15 – Decoding an MPS or an LPS

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 C.16. As long as the MPS sub-interval size A calculated as the first step in Figure C.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 C.2. If, however, the LPS sub-interval is larger, the conditional exchange occurred and an LPS occurred. D is set by inverting MPS(CX). 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 C.2. The probability estimation in the decoder needs to be identical to the probability estimation in the encoder.

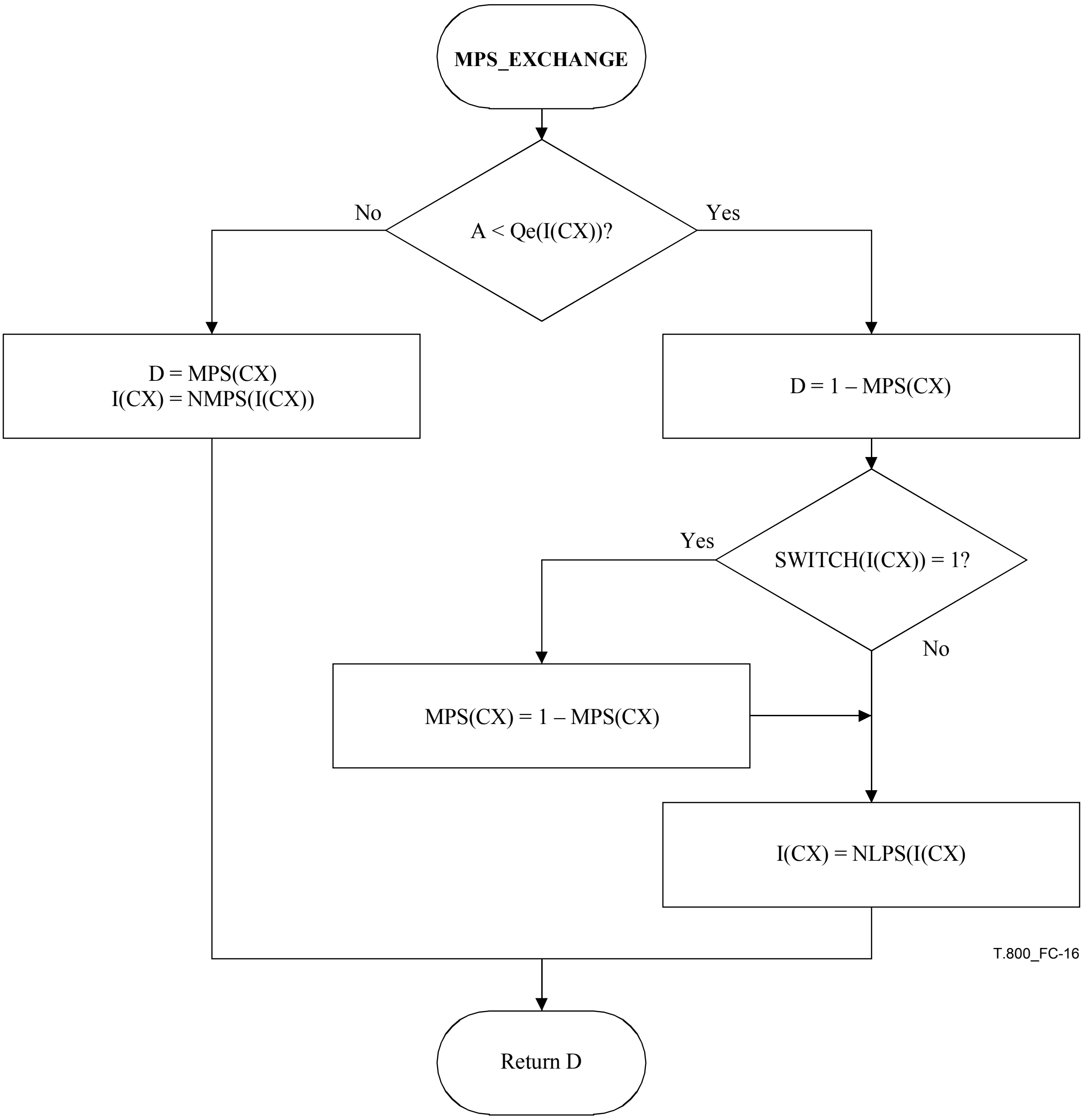


Figure C.16 – Decoder MPS path conditional exchange procedure

For the LPS path of the decoder, the conditional exchange procedure is given the LPS_EXCHANGE procedure shown in Figure C.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.

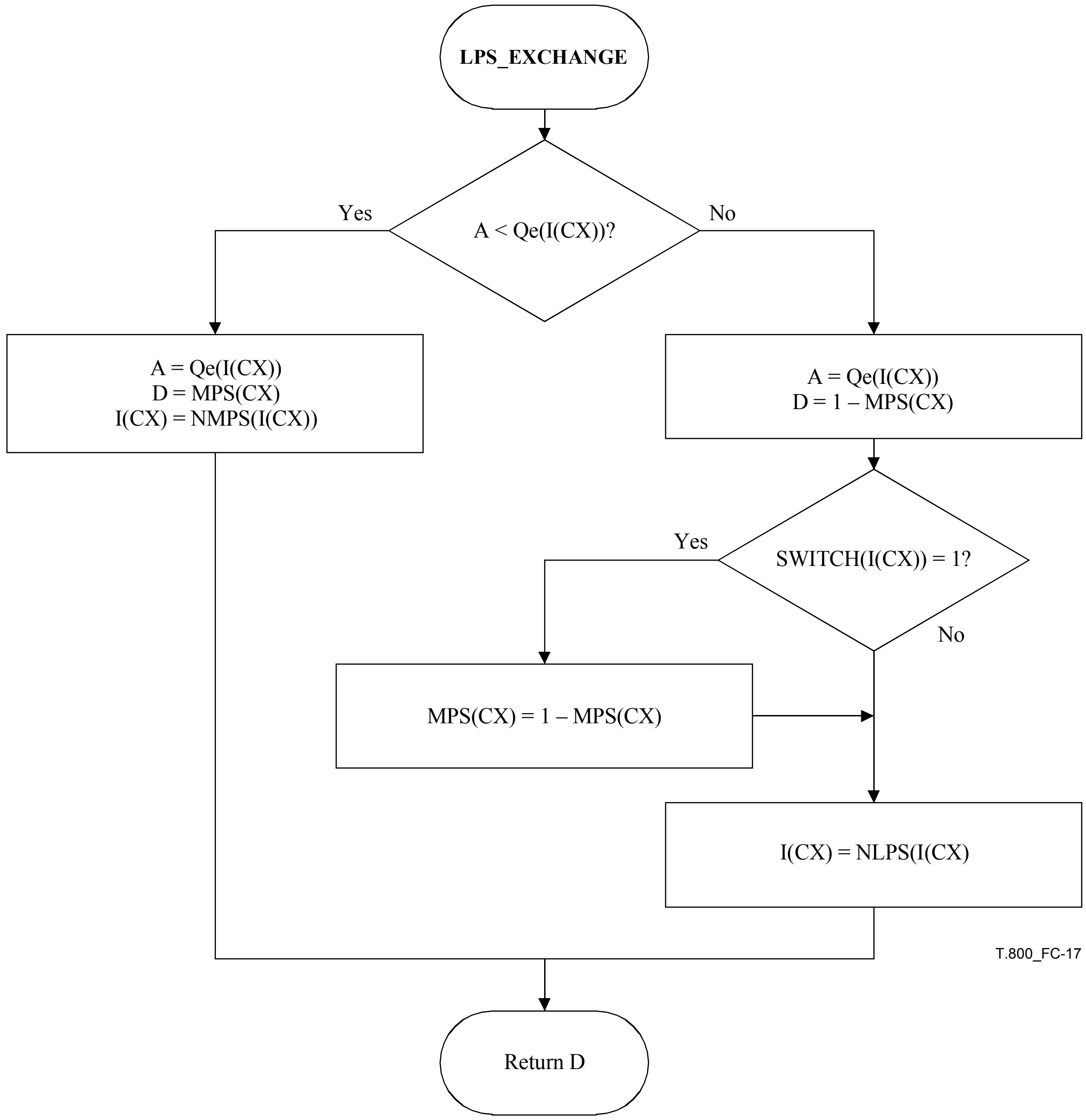


Figure C.17 – Decoder LPS path conditional exchange procedure

C.3.3 Renormalization in the decoder (RENORMD)

The RENORMD procedure for the decoder renormalization is illustrated in Figure C.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. The C-register in this procedure is the concatenation of the Chigh and Clow registers.

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.

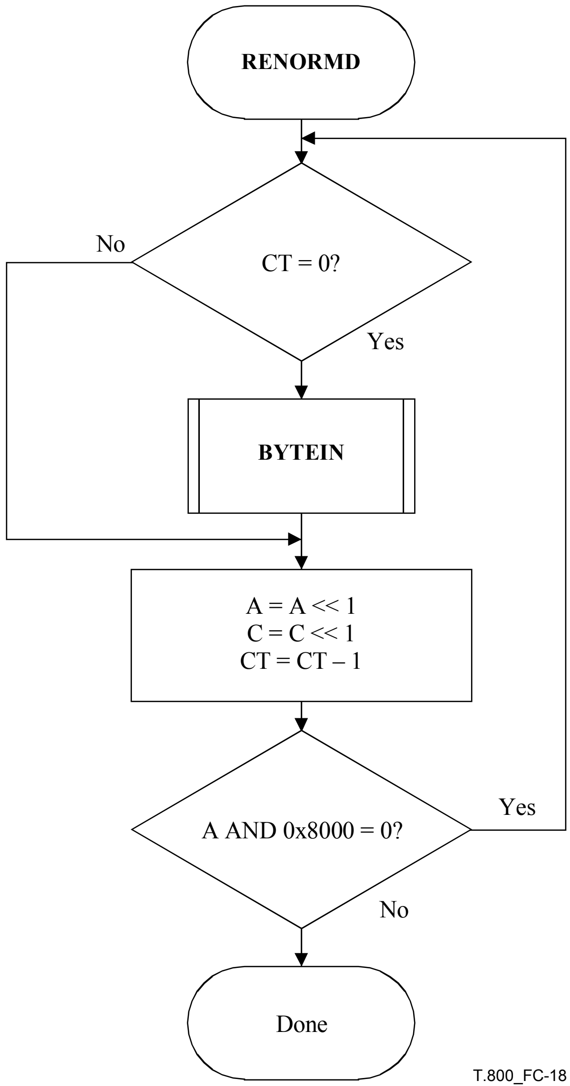


Figure C.18 – Decoder renormalization procedure

C.3.4 Compressed image data input (BYTEIN)

The BYTEIN procedure called from RENORMD is illustrated in Figure C.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 coding pass. The C-register in this procedure is the concatenation of the Chigh and Clow registers.

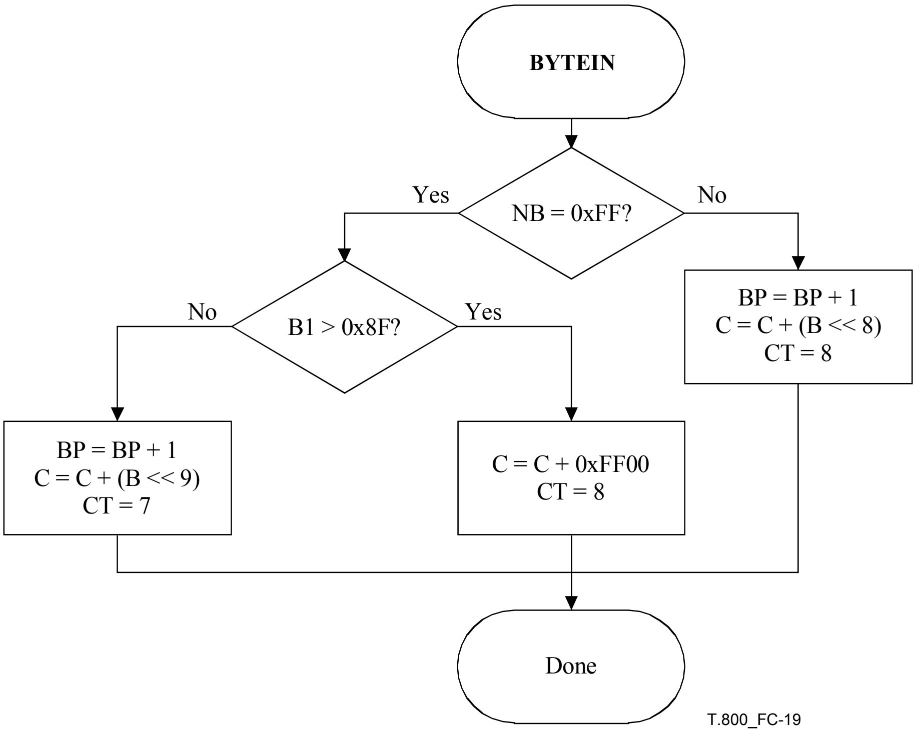


Figure C.19 – BYTEIN procedure for decoder

B is the byte pointed to by the compressed image 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 image 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.

C.3.5 Initialization of the decoder (INITDEC)

The INITDEC procedure is used to start the arithmetic decoder. After MPS and I are initialized, the basic steps are shown in Figure C.20.

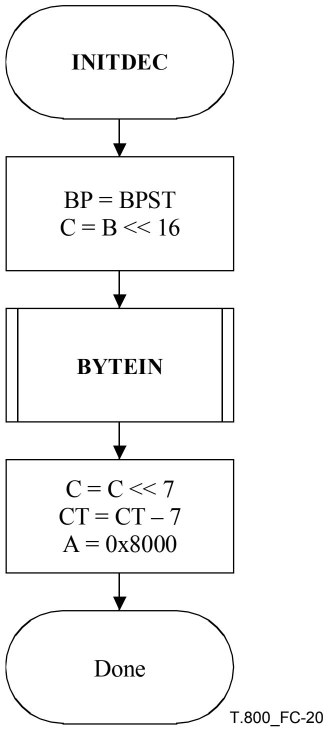


Figure C.20 – Initialization of the decoder

BP, the pointer to the compressed image data, is initialized to BPST (pointing to the first compressed byte). The first byte of the compressed image 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. The initial settings for MPS and I are shown in Table D.7.

C.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 returning I(CX) and MPS(CX) to their initial values as defined in Table D.7 for some or all values of CX.

C.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 D

Coefficient bit modeling

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

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

This annex defines the modeling and scanning of transform coefficient bits.

Code-blocks (see Annex B) are decoded a bit-plane at a time starting from 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 coding passes. Each coefficient bit in the bit-plane appears in only one of the three coding passes called significance propagation, magnitude refinement, and cleanup. For each pass contexts are created which are provided to the arithmetic coder, CX, along with the bit stream, CD (see C.3).

D.1 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 coefficients of the first column are scanned, followed by the first four coefficients of the second column and so on, until the right side of the code-block is reached. The scan then returns to the left of the code-block and the second set of four coefficients in each column is scanned. The process is continued to the bottom of the code-block. If the code-block height is not divisible by 4, the last set of coefficients scanned in each column will contain fewer than 4 members. Figure D.1 shows an example of the code-block scan pattern for a code-block.

Code-block 16 wide by N high

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	...														
...															

T.800_FD-1

Figure D.1 – Example scan pattern of a code-block bit-plane

D.2 Coefficient bits and significance

D.2.1 General case notations

The decoding procedures specified in this annex produce for each transform coefficient (u, v) of sub-band b the decoded bits which will be used to reconstruct the transform coefficient value $q_b(u, v)$. The bits produced are: a sign bit $s_b(u, v)$ and a number $N_b(u, v)$ of decoded magnitude MSBs, ordered from most to least significant: $MSB_i(b, u, v)$ is the ith MSB of transform coefficient (u, v) of sub-band b (i = 1, ..., $N_b(u, v)$). As indicated in Equation (D-1), the sign bit $s_b(u, v)$ has a value of one for negative coefficients and of zero for positive coefficients. The number $N_b(u, v)$ of decoded MSBs includes the number of all zero most significant bit-planes signalled in the packet header (see B.10.5).

D.2.2 Notation in the case with ROI

In the case of the presence of the RGN marker segment (indicating the presence of an ROI), modifications need to be made to the decoded bits, as well as the number of decoded bits $N_b(u, v)$. These modifications are specified in H.1. In the absence of the RGN marker segment, no modification is required.

D.3 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 decoding of the code-block. The "significance state" changes from insignificant to significant (see the clause below) at the bit-plane where the most significant magnitude bit equal to 1 is found. 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 D.2. 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 creating a context vector for decoding the current coefficient.

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

T.800_FD-2

Figure D.2 – Neighbors states used to form the context

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 passes: 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 pass, 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 first bit-plane within the current block with a non-zero element has a cleanup pass only. The remaining bit-planes are decoded in three coding passes. Each coefficient bit is decoded in exactly one of the three coding passes. Which pass a coefficient bit is decoded in depends on the conditions for that pass. In general, the significance propagation pass 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. The cleanup pass includes all the remaining coefficients.

D.3.1 Significance propagation decoding pass

The eight surrounding neighbor coefficients of a current coefficient (shown in Figure D.2 where X denotes the current coefficient) are used to create a context vector that maps into one of the 9 contexts shown in Table D.1. 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 the sub-band.

Table D.1 – Contexts for the significance propagation and cleanup coding passes

LL and LH sub-bands (vertical high-pass)			HL sub-band (horizontal high-pass)			HH sub-band (diagonally high-pass)		Context label[a]
$\sum H_i$	$\sum V_i$	$\sum D_i$	$\sum H_i$	$\sum V_i$	$\sum D_i$	$\sum (H_i + V_i)$	$\sum D_i$	
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 indexed only for identification convenience in this Recommendation | International Standard. The actual identifiers used is a matter of implementation.

b) "x" indicates a "don't care" state.

The significance propagation pass only includes bits of coefficients that were insignificant (the significance state has yet to be set) 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. If the value of 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 coding passes are considered, the most current significance state for this coefficient is used.

D.3.2 Sign bit decoding

The context label for sign bit decoding is determined using another context vector from 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 D.2) 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 the sign is positive or –1 if the sign is 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 D.2 shows these contributions.

Table D.2 – Contributions of the vertical (and the horizontal) neighbors to the sign context

V0 (or H0)	V1 (or H1)	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 D.3 shows these context labels. This context is provided to the arithmetic decoder with the bit stream. The bit returned, *D* (see Annex C), is then logically exclusive ORed with the *XORbit* in Table D.3 to produce the sign bit. The following equation is used:

$$signbit = D \otimes XORbit \qquad \text{(D-1)}$$

where *signbit* is the sign bit of the current coefficient (a one bit indicates a negative coefficient, a zero bit a positive coefficient), *D* is the value returned from the arithmetic decoder given the context label and the bit stream, and the *XORbit* is found in Table D.3 for the current context label.

Table D.3 – Sign contexts from the vertical and horizontal contributions

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

D.3.3 Magnitude refinement pass

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

The context used is determined by the summation of the significance state of the horizontal, vertical, and diagonal neighbors. These are the states as currently known to the decoder, not the states used before the significance decoding pass. Further, it is dependent on whether this is the first refinement bit (the bit immediately after the significance and sign bits) or not. Table D.4 shows the three contexts for this pass.

Table D.4 – Contexts for the magnitude refinement coding passes

$\sum H_i + \sum V_i + \sum D_i$	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.		

The context is passed to the arithmetic coder along with the bit stream. The bit returned is the value of the current coefficient in the current bit-plane.

D.3.4 Cleanup pass

The remaining coefficients were previously insignificant and not handled by the significance propagation pass. The cleanup pass not only uses the neighbor context, like that of the significance propagation pass, from Table D.1, but also a run-length context.

During this pass the neighbor contexts for the coefficients in this pass are recreated using Table D.1. The context label can now have any value because the coefficients that were found to be significant in the significance propagation pass are considered to be 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 decoded in the cleanup pass and the context label for all is 0 (including context coefficients from previous magnitude, significance and cleanup passes), 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 and are set to zero.

Otherwise, if the symbol 1 is returned, then at least one of the four contiguous coefficients in the column is significant. The next two bits, returned with the UNIFORM context (index 46 in Table C.2), denote which coefficient from the top of the column down is the first to be found significant. The two bits decoded with the UNIFORM context are decoded MSB then LSB. That coefficient's sign bit is determined as described in D.3.2. The decoding of any remaining coefficients continues in the manner described in D.3.1.

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 D.1 as in the significance propagation pass. The same contexts as the significance propagation are used here (the state is used as well as the model). Table D.5 shows the logic for the cleanup pass.

Table D.5 – Run-length decoder for cleanup passes

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

[a] See Annex C.

If there are fewer than four rows remaining in a code-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.

D.3.5 Example of coding passes and significance propagation (informative)

Table D.6 shows an example of the decoding order for the quantized coefficients of one 4-coefficient column in the scan. This example assumes all neighbors not included in the table are identically zero, and indicates in which pass each bit is decoded. The sign bit is decoded after the initial 1 bit and is indicated in the table by the + or – sign. The very first pass in a new block is always a cleanup pass because there can be no predicted significant, or refinement bits. After the first pass, the decoded 1 bit of the first coefficient causes the second coefficient to be decoded in the significance pass for the next bit-plane. The 1 bit decoded for the last coefficient in the second cleanup pass causes the third coefficient to be decoded in the next significance pass.

Table D.6 – Example of sub-bit-plane coding order and significance propagation

Coding passes	10	1	3	–7	Coefficient value
	+	+	+	–	Coefficient sign
	1	0	0	0	Coefficient magnitude (MSB to LSB)
	0	0	0	1	
	1	0	1	1	
	0	1	1	1	
Cleanup	1+	0	0	0	
Significance		0			
Refinement	0				
Cleanup			0	1–	
Significance		0	1+		
Refinement	1			1	
Cleanup					
Significance		1+			
Refinement	0		1	1	
Cleanup					

D.4 Initializing and terminating

When the contexts are initialized, or re-initialized, they are set to the values in the Table D.7.

Table D.7 – Initial states for all contexts

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

In normal operation (not selective arithmetic coding bypass), the arithmetic coder shall be terminated either at the end of every coding pass or only at the end of every code-block (see D.4.1). Table D.8 shows two examples of termination patterns for the coding passes in a code-block. The COD or COC marker signals which termination pattern is used (see A.6.1 and A.6.2).

Table D.8 – Arithmetic coder termination patterns

#	Pass	Coding Operation Termination only on last pass	Coding Operation Termination on every pass
1	cleanup	Arithmetic Coder (AC)	AC, terminate
2	significance propagation	AC	AC, terminate
2	magnitude refinement	AC	AC, terminate
2	cleanup	AC	AC, terminate
...	...	...	...
final	significance propagation	AC	AC, terminate
final	magnitude refinement	AC	AC, terminate
final	cleanup	AC, terminate	AC, terminate

When multiple terminations of the arithmetic coder are present, the length of each terminated segment is signalled in the packet header as described in B.10.7.

NOTE – Termination should never create a byte aligned value between 0xFF90 and 0xFFFF inclusive. These values are available as in-bit-stream marker values.

D.4.1 Expected codestream 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 bytes for those coding passes have been consumed. The number of bytes corresponding to the coding passes is specified in the packet header. 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 C.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 decoder.

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.

D.4.2 Arithmetic coder termination

The FLUSH procedure performs this task (see C.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.

When the predictable termination flag is set (see COD and COC in A.6.1 and A.6.2) the following termination procedure shall be used. Using the notation of C.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) + 1.
2) While (k > 0):
   - shift C left by CT and set CT = 0.
   - execute the BYTEOUT procedure. 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. 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.

If the predictable termination flag is not set, 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.

D.4.3 Length computation (informative)

To include coding pass compressed image data into packets, the number of bytes to be included must be determined. If the coding pass compressed image data is terminated, the algorithm in the previous clause may be used. Otherwise, the encoder should calculate a suitable length such that corresponding bytes are sufficient for the decoder to reconstruct the coding passes.

D.5 Error resilience segmentation symbol

A segmentation symbol is a special symbol. Whether it is used is signalled in the COD or COC marker segments (see A.6.1 and A.6.2). The symbol is coded with the UNIFORM context of the arithmetic coder at the end of each bit-plane. The correct decoding of this symbol confirms the correctness of the decoding of this bit-plane, which allows error detection. At the decoder, a segmentation symbol 1010 or 0xA should be decoded at the end of each bit-plane (at the end of a cleanup pass). If the segmentation symbol is not decoded correctly, then bit errors occurred for this bit-plane.

NOTE – This can be used with or without the predictable termination.

D.6 Selective arithmetic coding bypass

This style of coding allows bypassing the arithmetic coder for the significance propagation pass and magnitude refinement coding passes starting in the fifth significant bit-plane of the code-block. The COD or COC marker signals whether or not this coding style is used (see A.6.1 and A.6.2).

The first cleanup pass (which is the first bit-plane of a code-block with a non-zero element) and the next three sets of significance propagation, magnitude refinement, and cleanup coding passes are decoded with the arithmetic coder. The fourth cleanup pass shall include an arithmetic coder termination (see Table D.9).

Table D.9 – Selective arithmetic coding bypass

Bit-plane number	Pass type	Coding operations
1	cleanup	Arithmetic Coding (AC)
2	significance propagation	AC
2	magnitude refinement	AC
2	cleanup	AC
3	significance propagation	AC
3	magnitude refinement	AC
3	cleanup	AC
4	significance propagation	AC
4	magnitude refinement	AC
4	cleanup	AC, terminate
5	significance propagation	raw
5	magnitude refinement	raw, terminate
5	cleanup	AC, terminate
...	...	...
final	significance	raw
final	magnitude refinement	raw, terminate
final	cleanup	AC, terminate

Starting with the fourth significance propagation and magnitude refinement coding passes, the bits that would have been returned from the arithmetic coder are instead returned directly from the bit stream. (A routine that undoes the effects of bit stuffing precedes the return of bits. Specifically, this routine throws out the first bit after an 0xFF byte value.) After each magnitude refinement pass the bit stream has been "terminated" by padding to the byte boundary.

When the predictable termination flag is set (see COD and COC in A.6.1 and A.6.2) and all the bits from a magnitude refinement pass have been assembled, any remaining bits in the last byte are filled with an alternating sequence of 0s and 1s. This sequence should start with a 0 regardless of the number of bits to be padded.

When the termination on each coding pass flag is set (see COD and COC in A.6.1 and A.6.2), then the significance propagation passes are terminated in the same way as the magnitude refinement passes.

The cleanup coding passes continue to receive compressed image data directly from the arithmetic coder and are always terminated.

The sign bit is computed with Equation (D-2):

$$signbit = raw\_value \tag{D-2}$$

where $raw\_value = 1$ is a negative sign bit and $raw\_value = 0$ is a positive sign bit. Table D.9 shows the coding sequence.

The length of each terminated segment, plus the length of any remaining unterminated passes, is signalled in the packet header as described in B.10.7. If termination on each coding pass is selected (see A.6.1 and A.6.2), then every pass is terminated (including both raw passes).

NOTE 1 – Using the selective bypass mode when encoding an image with an ROI may significantly decrease the compression efficiency.

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 2 – Since the decoder appends 0xFF values, as necessary, to the bit stream representing the coding pass (see D.4.1), truncation of the bit stream may be possible. When the predictable termination flag is set (see COD and COC in A.6.1 and A.6.2), such truncation is not permitted. The last byte cannot be an 0xFF, since the bit-stuffing routine appends a new byte following the FF, having most significant bit value of 0 and unused bits filled with the alternating sequence of 0 and 1 value bits.

D.7 Vertically causal context formation

This style of coding constrains the context formation to the current and past code-block scans (four rows of vertically scanned coefficients). That is, any coefficient from the next code-block scan are considered to be insignificant. The COD or COC marker signals whether or not this style of coding is used (see A.6.1 and A.6.2).

To illustrate, the bit labelled 14 in Figure D.1 is decoded as usual using the neighbor states as specified in Figure D.2, independent of whether or not contexts are vertically causal. However when vertically causal context formation is used, the bit labeled 15 is decoded assuming $D_2 = V_1 = D_3 = 0$ in Figure D.2.

D.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 D.3. The decisions made are in Table D.10 and the bits and context sent to the coder are in Table D.11. These show the context without the selective arithmetic coding bypass or the vertically causal model.

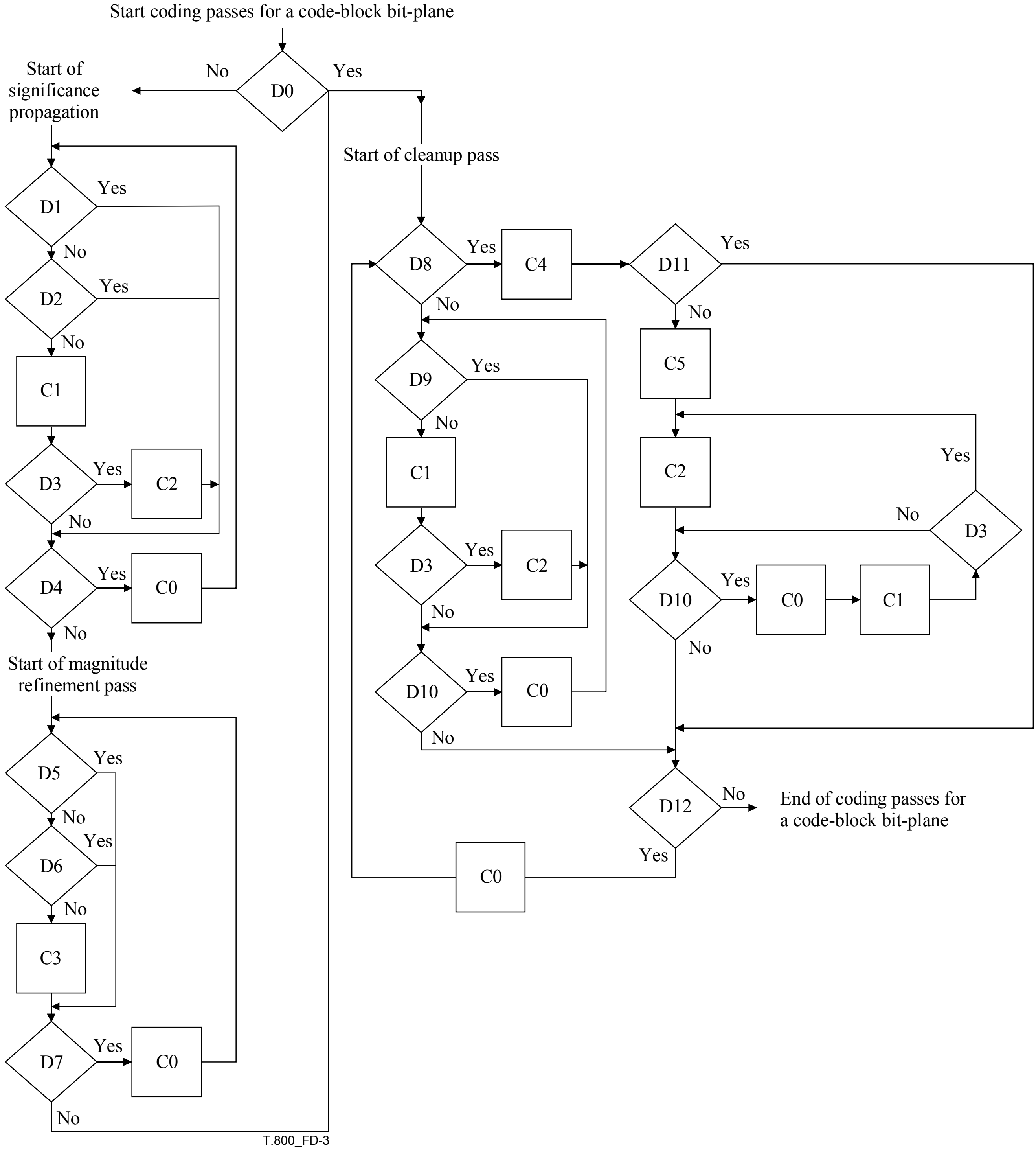


Figure D.3 – Flow chart for all coding passes on a code-block bit-plane

Table D.10 – Decisions in the context model flow chart

Decision	Question	Description
D0	Is this the first significant bit-plane for the code-block?	See D.3
D1	Is the current coefficient significant?	See D.3.1
D2	Is the context bin zero? (see Table D.1)	See D.3.1
D3	Did the current coefficient just become significant?	See D.3.1
D4	Are there more coefficients in the significance propagation?	
D5	Is the coefficient insignificant?	See D.3.3
D6	Was the coefficient coded in the last significance propagation?	See D.3.3
D7	Are there more coefficients in the magnitude refinement pass?	
D8	Are four contiguous undecoded coefficients in a column each with a 0 context?	See D.3.4
D9	Is the coefficient significant or has the bit already been coded during the Significance Propagation coding pass?	See D.3.4
D10	Are there more coefficients remaining of the four column coefficients?	
D11	Are the four contiguous bits all zero?	See D.3.4
D12	Are there more coefficients in the cleanup pass?	

Table D.11 – Decoding in the context model flow chart

Code	Decoded symbol	Context	Brief explanation	Description
C0	–	–	Go to the next coefficient or column	
C1	Newly significant?	Table D.1, 9 context labels	Decode significance bit of current coefficient (significance propagation or cleanup)	See D.3.1
C2	Sign bit	Table D.3, 5 context labels	Decode sign bit of current coefficient	See D.3.2
C3	Current magnitude bit	Table D.4, 3 context labels	Decode magnitude refinement pass bit of current coefficient	See D.3.3
C4	0 1	Run-length context label	Decode run-length of four zeros Decode run-length not of four zeros	See D.3.4
C5	00 01 10 11	UNIFORM	First coefficient is first with non-zero bit Second coefficient is first with non-zero bit Third coefficient is first with non-zero bit Fourth coefficient is first with non-zero bit	See D.3.4 and Table C.2

Annex E

Quantization

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

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

This annex specifies the forms of inverse quantization used for the reconstruction of tile-component transform coefficients. Information about quantization of transform coefficients for encoding is also provided. Quantization is the process by which the transform coefficients are reduced in precision.

E.1 Inverse quantization procedure

For each transform coefficient (u, v) of a given sub-band b, the transform coefficient value $q_b(u, v)$ is given by the following equation:

$$\overline{q_b}(u,v) = (1 - 2s_b(u,v)) \cdot \left(\sum_{i=1}^{N_b(u,v)} MSB_i(b,u,v) \cdot 2^{M_b - i} \right) \qquad \text{(E-1)}$$

where $s_b(u, v)$, $N_b(u, v)$ and $MSB_i(b, u, v)$are given in D.2, and where M_b is retrieved using Equation (E-2), where the number of guard bits G and the exponent ε_b are specified in the QCD or QCC marker segments (see A.6.4 and A.6.5).

$$M_b = G + \varepsilon_b - 1 \qquad \text{(E-2)}$$

Each decoded transform coefficient $q_b(u, v)$ of sub-band b is used to generate a reconstructed transform coefficient $Rq_b(u, v)$, as will be described in E.1.1.

NOTE – Decoding only $N_b(u, v)$ (see D.2.1) bit-planes is equivalent to decoding data which has been encoded using a scalar quantizer with step size $2^{M_b - N_b(u,v)} \cdot \Delta_b$ for all the coefficients of this code-block. Due to the nature of the three coding passes (see D.3), $N_b(u, v)$ may be different for different coefficients within the same code-block.

E.1.1 Irreversible transformation

E.1.1.1 Determination of the quantization step size

For the irreversible transformation, the quantization step size Δ_b for a given sub-band b is calculated from the dynamic range R_b of sub-band b, the exponent ε_b and mantissa μ_b as given in Equation (E-3).

$$\Delta_b = 2^{R_b - \varepsilon_b} \left(1 + \frac{\mu_b}{2^{11}} \right) \qquad \text{(E-3)}$$

NOTE – The denominator, 2^{11}, in Equation (E-3) is due to the allocation of 11 bits in the codestream for μ_b, as given in Table A.30.

In Equation (E-3), the exponent ε_b and the mantissa μ_b are specified in the QCD or QCC marker segments (see A.6.4 and A.6.5), and the nominal dynamic range R_b (as given by Equation (E-4)) is the sum of R_I (the number of bits used to represent the original tile-component samples which can be extracted from the SIZ marker – see Table A.11 in A.5.1) and the base 2 exponent of the sub-band gain (*gain*$_b$) of the current sub-band b, which varies with the type of sub-band b (*lev*LL, *lev*LH or *lev*HL, *lev*HH – see F.3.1) and can be found in Table E.1.

Table E.1 – Sub-band gains

sub-band b	$gain_b$	$\log_2(gain_b)$
*lev*LL	1	0
*lev*LH	2	1
*lev*HL	2	1
*lev*HH	4	2

$$R_b = R_I + \log_2\left(gain_b\right) \tag{E-4}$$

The exponent/mantissa pairs (ε_b,μ_b) are either signalled in the codestream for every sub-band (expounded quantization) or else signalled only for the N_LLL sub-band and derived for all other sub-bands (derived quantization) (see Table A.30). In the case of derived quantization, all exponent/mantissa pairs (ε_b,μ_b) are derived from the single exponent/mantissa pair (ε_o,μ_o) corresponding to the N_LLL sub-band, according to Equation (E-5):

$$\left(\varepsilon_b, \mu_b\right) = \left(\varepsilon_o - N_L + n_b, \mu_o\right) \tag{E-5}$$

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

NOTE – For a given sub-band b, a quantized transform coefficient may exceed the dynamic range R_b.

E.1.1.2 Reconstruction of the transform coefficient

For the irreversible transformation, the reconstructed transform coefficient is given by Equation (E-6):

$$Rq_b(u,v) = \begin{cases} \left(\overline{q_b}(u,v) + r2^{M_b - N_b(u,v)}\right) \cdot \Delta_b & \text{for } \overline{q_b}(u,v) > 0 \\ \left(\overline{q_b}(u,v) - r2^{M_b - N_b(u,v)}\right) \cdot \Delta_b & \text{for } \overline{q_b}(u,v) < 0 \\ 0 & \text{for } \overline{q_b}(u,v) = 0 \end{cases} \tag{E-6}$$

where r is a reconstruction parameter, which can be arbitrarily chosen by the decoder.

NOTE – The reconstruction parameter r may be chosen for example to produce the best visual or objective quality for reconstruction. Generally, values for r fall in the range of $0 \le r < 1$, and a common value is $r = 1/2$. (This note also applies to E.1.2).

E.1.2 Reversible transformation

E.1.2.1 Determination of the quantization step size

For the reversible transformation, the quantization step size Δ_b is equal to one (no quantization is performed).

E.1.2.2 Reconstruction of the transform coefficient

For the reversible transformation, the reconstructed transform coefficient $Rq_b(u, v)$ is recovered differently depending on whether all the coefficient bits are decoded, i.e., whether $N_b(u, v) = M_b$ or $N_b(u, v) < M_b$.

If $N_b(u, v) = M_b$, then the reconstructed transform coefficient $Rq_b(u, v)$ is given by Equation (E-7).

$$Rq_b(u,v) = \overline{q_b}(u,v) \tag{E-7}$$

If $N_b(u, v) < M_b$, then the reconstructed transform coefficient $Rq_b(u, v)$ is given by Equation (E-8).

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

E.2 Scalar coefficient quantization (informative)

For irreversible compression, after the irreversible forward discrete wavelet transformation (see Annex F), each of the transform coefficients $a_b(u, v)$ of the sub-band is quantized to the value $q_b(u, v)$ according to Equation (E-9).

$$q_b(u,v) = sign(a_b(u,v)) \cdot \left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor \quad \text{(E-9)}$$

where Δ_b is the quantization step size. The exponent ε_b and mantissa corresponding to Δ_b can be derived from Equation (E-5), and must be recorded in the codestream in the QCD or QCC markers (see A.6.4 and A.6.5).

For reversible compression, the quantization step size is required to be 1. In this case, a parameter ε_b has to be recorded in the codestream in the QCD or QCC markers (see A.6.4 and A.6.5), and is calculated as:

$$\varepsilon_b = R_I + log_2(gain_b) + \zeta_c \quad \text{(E-10)}$$

where R_I and $gain_b$ are as described in E.1.1, and where ζ_c is zero if the RCT is not used and ζ_c is the number of additional bits added by the RCT if the RCT is used, as described in G.2.1.

For both reversible and irreversible compression, 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 M_b of bits for the integer representation of $q_b(u, v)$ used at the encoder side is defined by Equation (E-2). The number G of guard bits has to be specified in the QCD or QCC marker (see A.6.4 and A.6.5). Typical values for the number of guard bits are $G = 1$ or $G = 2$.

Annex F

Discrete wavelet transformation of tile-components

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

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

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

F.1 Tile-component parameters

Consider the tile-component defined by the coordinates, tcx_0, tcx_1, tcy_0 and tcy_1 given in Equation (B-12), in Annex B.3. Then the coordinates (x, y) of the tile-component (with sample values $I(x, y)$ lie in the range defined by:

$$tcx_0 \le x < tcx_1 \text{ and } tcy_0 \le y < tcy_1 \tag{F-1}$$

F.2 Discrete wavelet transformations

F.2.1 Low-pass and high-pass filtering (informative)

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

To perform the inverse discrete wavelet transformation (IDWT), this Recommendation | International Standard uses a one-dimensional sub-band reconstruction of a one-dimensional array of samples from low-pass and high-pass coefficients.

F.2.2 Decomposition levels

Each tile-component is transformed into a set of two-dimensional sub-band signals (called sub-bands), each representing the activity of the signal in various frequency bands, at various spatial resolutions. N_L denotes the number of decomposition levels.

F.2.3 Discrete wavelet filters (informative)

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

F.3 Inverse discrete wavelet transformation

F.3.1 The IDWT procedure

The inverse discrete wavelet transformation (IDWT) transforms a set of sub-bands, $a_b(u_b, v_b)$ into a DC-level shifted tile-component, $I(x, y)$ (IDWT procedure). The IDWT procedure also takes as input a parameter N_L, which represents the number of decomposition levels (see Figure F.1). The number of decomposition levels N_L is signalled in the COD or COC markers (see A.6.1 and A.6.2).

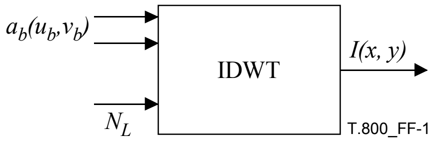


Figure F.1 – Inputs and outputs of the IDWT procedure

The sub-bands are labelled in the following way: an index *lev* corresponding to the decomposition level, followed by two letters which are either LL, HL, LH or HH.

The sub-band $b = lev$LL corresponds to a downsampled version of sub-band (lev – 1)LL which has been low-pass filtered vertically and low-pass filtered horizontally. The sub-band $b = 0$LL corresponds to the original tile-component. The sub-band $b = lev$HL corresponds to a downsampled version of sub-band (lev – 1)LL which has been low-pass filtered vertically and high-pass filtered horizontally. The sub-band $b = lev$LH corresponds to a downsampled version of sub-band (lev – 1)LL which has been high-pass filtered vertically and low-pass filtered horizontally. The sub-band $b = lev$HH corresponds to a downsampled version of sub-band (lev – 1)LL which has been high-pass filtered vertically and high-pass filtered horizontally.

For a given value of N_L, only the following sub-bands are present in the codestream, and in the following order (these sub-bands are sufficient to fully reconstruct the original tile-component):

N_LLL, N_LHL, N_LLH, N_LHH, (N_L – 1)HL, (N_L – 1)LH, (N_L – 1)HH,..., 1HL, 1LH, 1HH.

For a given sub-band b, the number n_b represents the decomposition level at which it has been generated at the time of encoding, and is given in Table F.1:

Table F.1 – Decomposition level n_b for sub-band b

b	N_LLL	N_LHL	N_LLH	N_LHH	(N_L – 1)HL	(N_L – 1)HL	(N_L – 1)HL	...	1HL	1LH	1HH
n_b	N_L	N_L	N_L	N_L	N_L – 1	N_L – 1	N_L – 1	...	1	1	1

The sub-bands for the case where N_L = 2 are illustrated in Figure F.2.

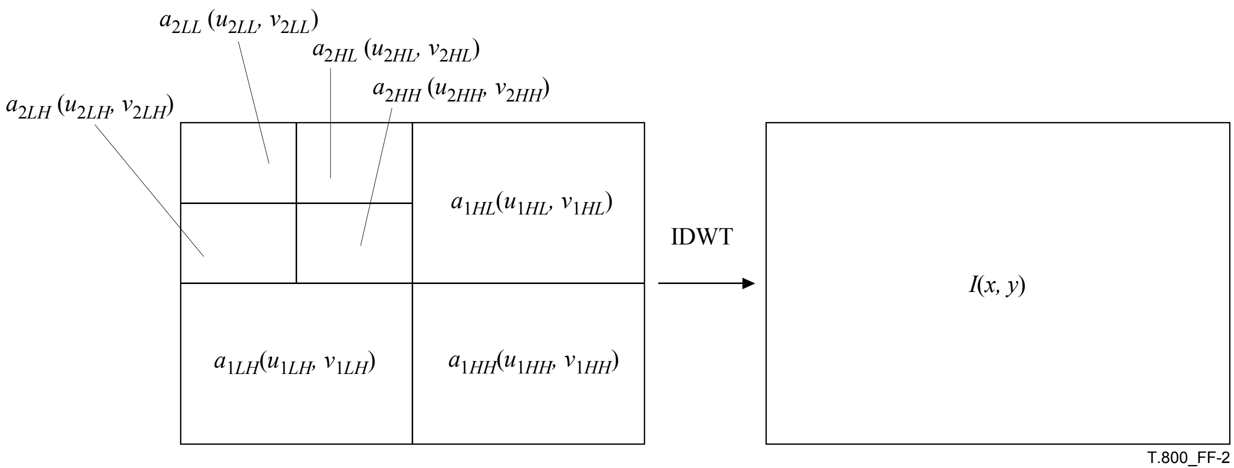


Figure F.2 – The IDWT (N_L = 2)

The IDWT procedure starts with the initialization of the variable *lev* (the current decomposition level) to N_L. The 2D_SR procedure (see F.3.2) is performed at every level *lev*, where the level *lev* decreases at each iteration, until N_L iterations are performed. The 2D_SR procedure is iterated over the *lev*LL sub-band produced at each iteration. Finally, the sub-band a_{0LL} (u_{0LL}, v_{0LL}) is the output array $I(x, y)$.

As defined in Equation (B-15), the indices (u_b, v_b) of sub-band coefficients $a_b(u_b, v_b)$ for a given sub-band b lie in the range defined by:

$$tbx_0 \le u_b < tbx_1 \quad \text{and} \quad tby_0 \le v_b < tby_1 \tag{F-2}$$

Figure F.3 describes the IDWT procedure.

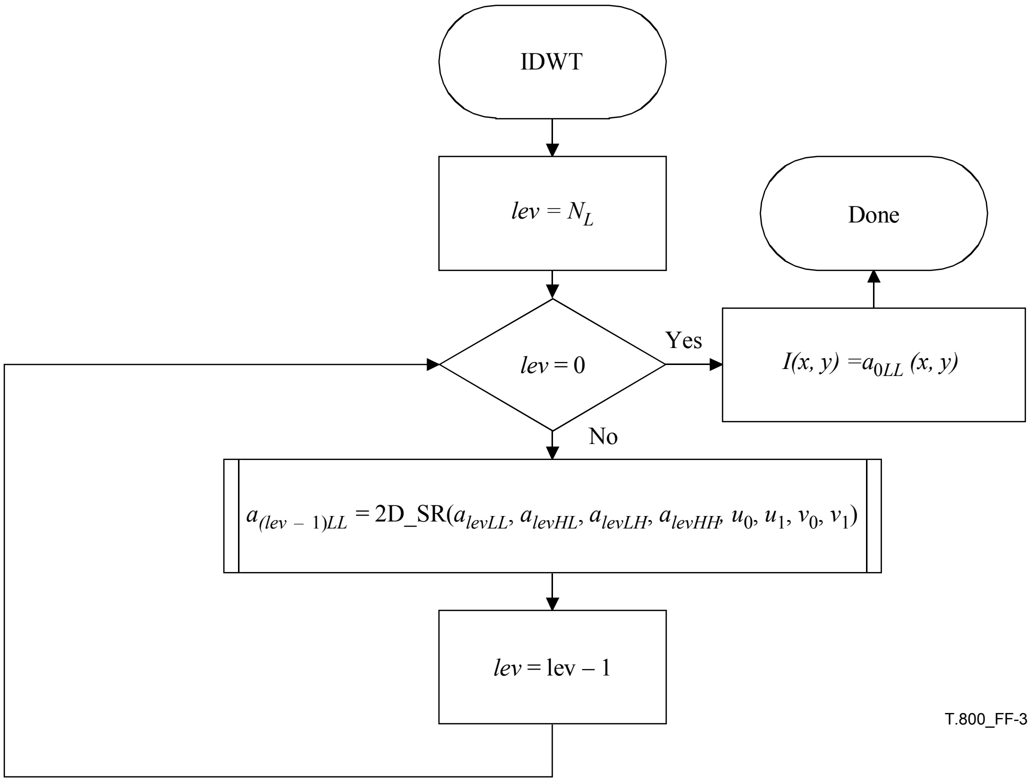


Figure F.3 – The IDWT procedure

F.3.2 The 2D_SR procedure

The 2D_SR procedure performs a reconstruction of sub-band $a_{(lev-1)LL}(u, v)$ from the four sub-bands, $a_{levLL}(u, v)$, $a_{levHL}(u, v)$, $a_{levLH}(u, v)$ and $a_{levHH}(u, v)$ (see Figure F.4). The total number of coefficients of the reconstructed *lev*LL sub-band is equal to the sum of the total number of coefficients of the four sub-bands input to the 2D_SR procedure (see Figure F.5).

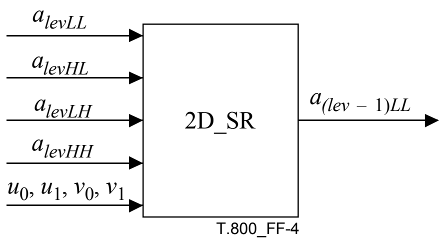


Figure F.4 – Inputs and outputs of the 2D_SR procedure

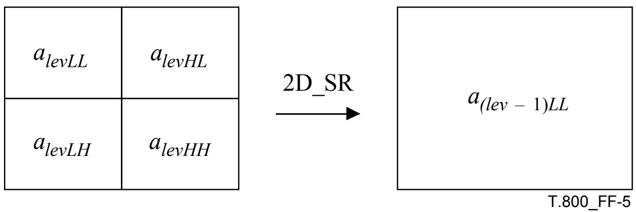


Figure F.5 – One level of reconstruction from four sub-bands (2D_SR procedure) into sub-bands

First, the four sub-bands $a_{levLL}(u, v)$, $a_{levHL}(u, v)$, $a_{levLH}(u, v)$ and $a_{levHH}(u, v)$ are interleaved to form an array $a(u, v)$ using the 2D_INTERLEAVE procedure. The 2D_SR procedure then applies the HOR_SR procedure to all rows of $a(u, v)$, and finally applies the VER_SR procedure to all columns of $a(u, v)$ to produce the reconstructed sub-band $a_{(lev-1)LL}(u, v)$. Figure F.6 describes the 2D_SR procedure.

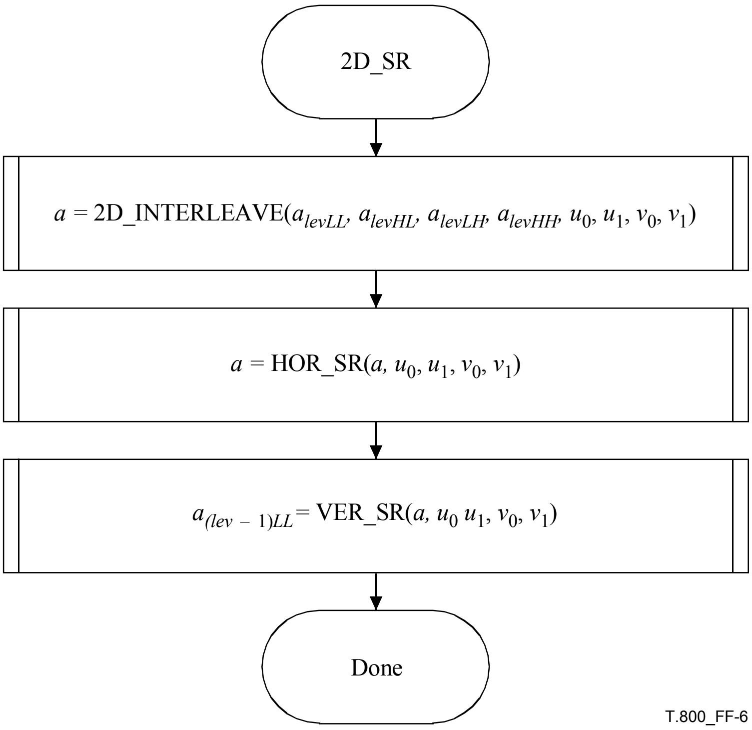


Figure F.6 – The 2D_SR procedure

F.3.3 The 2D_INTERLEAVE procedure

As illustrated in Figure F.7, the 2D_INTERLEAVE procedure interleaves the coefficients of four sub-bands a_{levLL}, a_{levHL}, a_{levLH}, a_{levHH} to form $a(u, v)$. The values of u_0, u_1, v_0, v_1 used by the 2D_INTERLEAVE procedure are those of tbx_0, tbx_1, tby_0, tby_1 of corresponding to sub-band $b = (lev - 1)$LL (see definition in Equation (B-15)).

The way these sub-bands are interleaved to form the output $a(u, v)$ is described by the 2D_INTERLEAVE procedure given in Figure F.8.

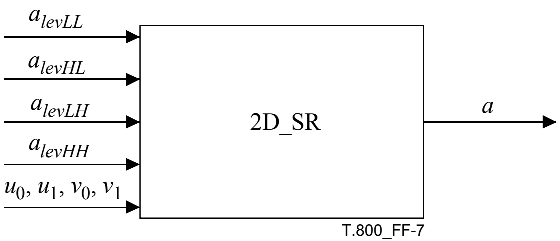


Figure F.7 – Parameters of 2D_INTERLEAVE procedure

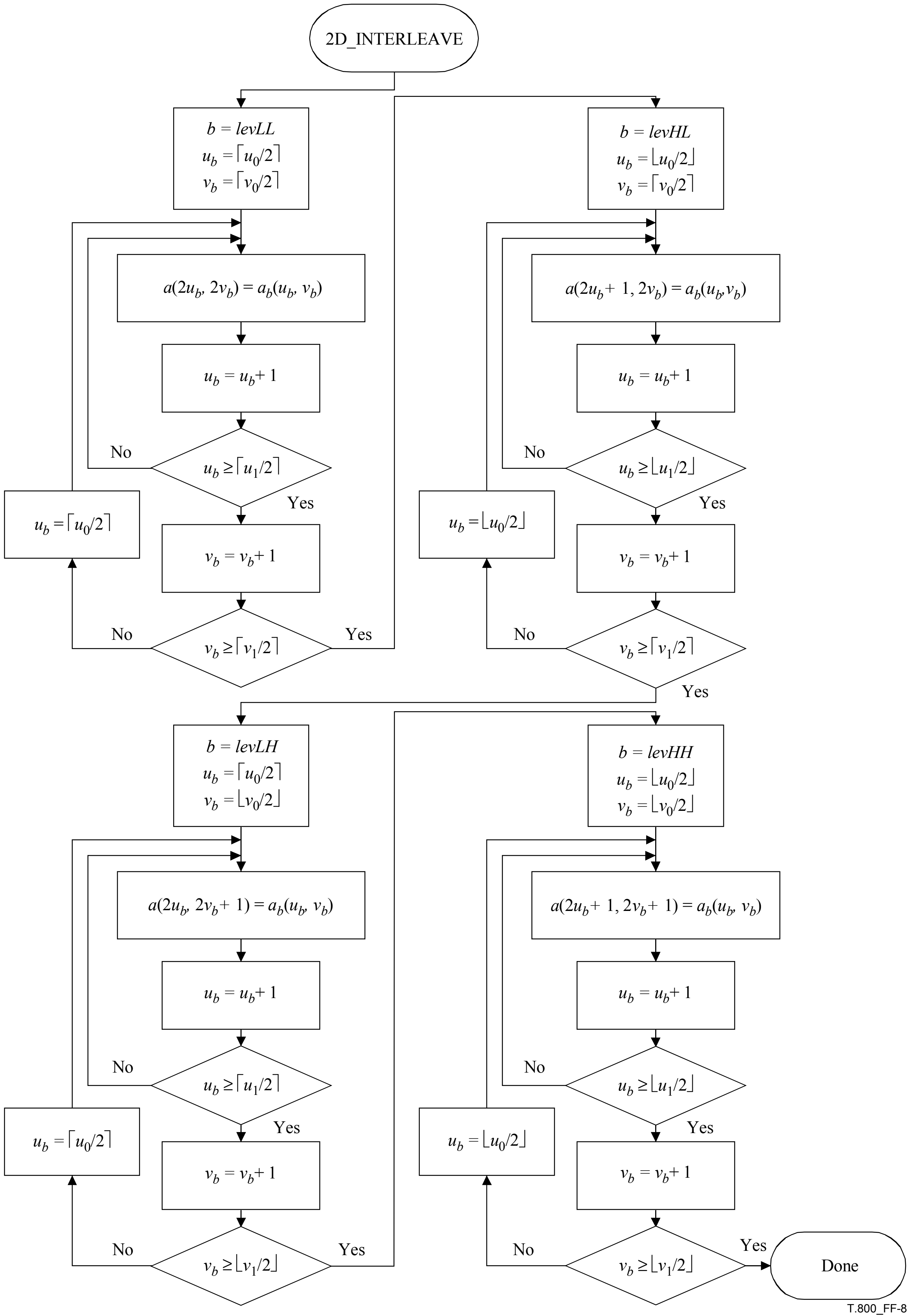


Figure F.8 – The 2D_INTERLEAVE procedure

F.3.4 The HOR_SR procedure

The HOR_SR procedure performs a horizontal sub-band reconstruction of a two-dimensional array of coefficients. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical extent of its coefficients as indicated by $u_0 \le u < u_1$ and $v_0 \le v < v_1$ (see Figure F.9) and produces as output a horizontally filtered version of the input array, row by row.

As illustrated in Figure F.10, the HOR_SR procedure applies the one-dimensional sub-band reconstruction (1D_SR procedure) to each row v of the input array $a(u, v)$, and stores the result back in each row.

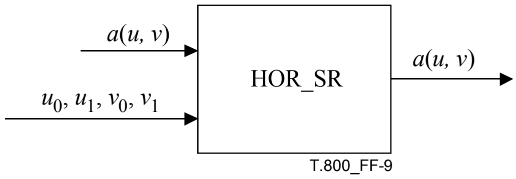


Figure F.9 – Inputs and outputs of the HOR_SR procedure

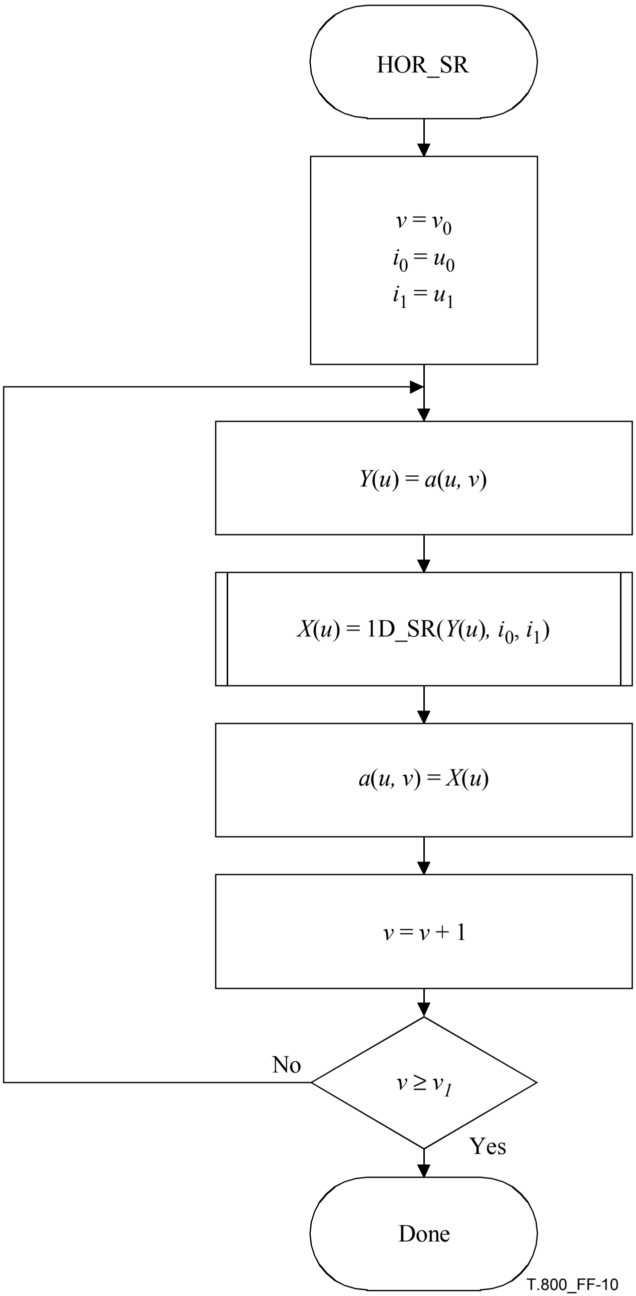


Figure F.10 – The HOR_SR procedure

F.3.5 The VER_SR procedure

The VER_SR procedure performs a vertical sub-band reconstruction of a two-dimensional array of coefficients. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical extent of its coefficients as indicated by $u_0 \le u < u_1$ and $v_0 \le v < v_1$ (see Figure F.11) and produces as output a vertically filtered version of the input array, column by column.

As illustrated in Figure F.12, the VER_SR procedure applies the one-dimensional sub-band reconstruction (1D_SR procedure) to each column u of the input array $a(u, v)$ and stores the result back in each column.

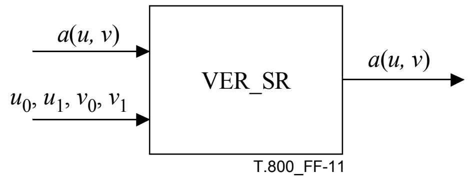


Figure F.11 – Inputs and outputs of the VER_SR procedure

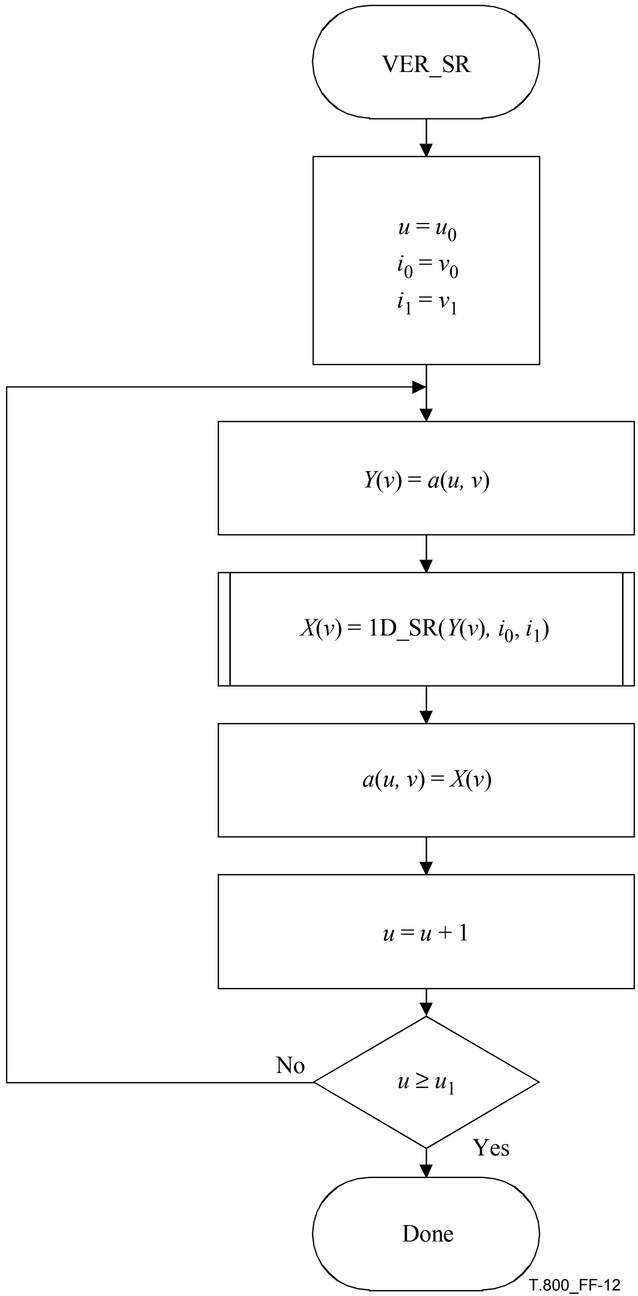


Figure F.12 – The VER_SR procedure

F.3.6 The 1D_SR procedure

As illustrated in Figure F.13, the 1D_SR procedure takes as input a one-dimensional array $Y(i)$, the extent of its coefficients as indicated by $i_0 \le i < i_1$. It produces as output an array X, with the same indices (i_0, i_1).

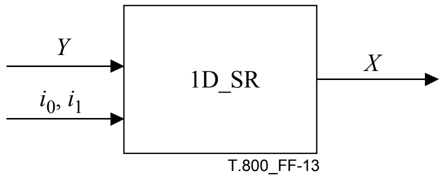


Figure F.13 – Parameters of the 1D_SR procedure

For signals of length one (i.e., $i_0 = i_1 - 1$), the 1D_SR procedure sets the value of $X(i_0)$ to $X(i_0)$ if i_0 is an even integer, and to $X(i_0) = Y(i_0)/2$ if i_0 is an odd integer.

For signals of length greater than or equal to two (i.e., $i_0 < i_1 - 1$), as illustrated in Figure F.14, the 1D_SR procedure first uses the 1D_EXTR procedure to extend the signal Y beyond its left and right boundaries resulting in the extended signal Y_{ext}, and then uses the 1D_FILTR procedure to inverse filter the extended signal Y_{ext} and produce the desired filtered signal X. The 1D_EXTR and 1D_FILTR procedures depend on whether the 9-7 irreversible wavelet transform (irreversible transformation) or 5-3 reversible wavelet transform (reversible transformation) is selected: this is signalled in the COD or COC markers (see A.6.1 and A.6.2).

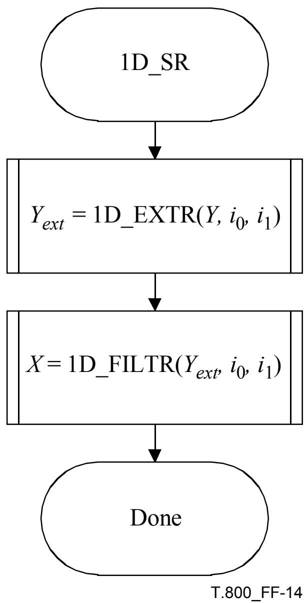


Figure F.14 – The 1D_SR procedure

F.3.7 The 1D_EXTR procedure

As illustrated in Figure F.15, the 1D_EXTR procedure extends signal Y by i_{left} coefficients to the left and i_{right} coefficients to the right. The extension of the signal is needed to enable filtering at both boundaries of the signal.

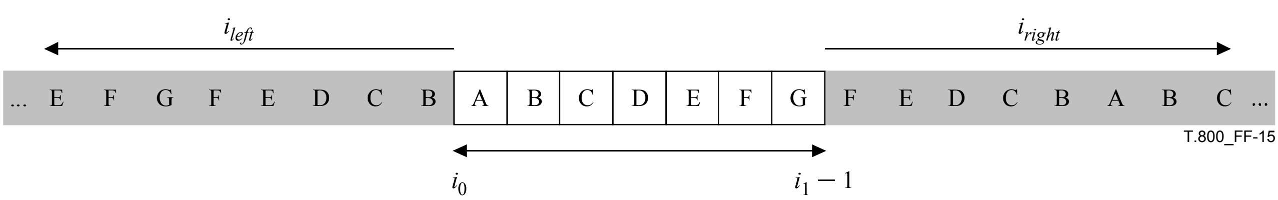


Figure F.15 – Periodic symmetric extension of signal

The first coefficient of Y is coefficient i_0, and the last coefficient of signal Y is coefficient $i_1 - 1$. This extension procedure is known as "periodic symmetric extension". Symmetric extension consists in extending the signal with the signal coefficients obtained by a reflection of the signal centered on the first coefficient (coefficient i_0) for extension to the left, and in extending the signal with the signal coefficients obtained by a reflection of the signal centred on the last coefficient (coefficient $i_1 - 1$) for extension to the right. Periodic symmetric extension is a generalization of symmetric extension for the more general case where the number of coefficients by which to extend the signal on any one side may exceed the signal length $i_1 - i_0$: this case may happen at higher decomposition levels.

The 1D_EXTR procedure calculates the values of $Y_{ext}(i)$ for values of i beyond the range $i_0 \le i < i_1$, as given in Equation (F-3):

$$Y_{ext}(i) = Y(PSE_O(i, i_0, i_1)) \tag{F-3}$$

where $PSE_O(i, i_0, i_1)$ is given by Equation (F-4):

$$PSE_O(i, i_0, i_1) = i_0 + \min(\mathrm{mod}(i - i_0, 2(i_1 - i_0 - 1)), 2(i_1 - i_0 - 1) - \mathrm{mod}(i - i_0, 2(i_1 - i_0 - 1))) \tag{F-4}$$

Two extension procedures are defined, depending on whether the 5-3 wavelet transformation ($1D\_EXTR_{5\text{-}3}$ procedure) or 9-7 wavelet transformation ($1D\_EXTR_{9\text{-}7}$ procedure). The procedures only differ in the minimum values of the extension parameters ($i_{left_{5\text{-}3}}$ and $i_{right_{5\text{-}3}}$ for the 5-3 wavelet transformation, and $i_{left_{9\text{-}7}}$ and $i_{right_{9\text{-}7}}$ for the 9-7 wavelet transformation) which are given in Tables F.2 and F.3, and depend on the parity of the indices i_0 and i_1. Values equal to or greater than those given in Tables F.2 and F.3 will produce the same array X at the output of the 1D_IFILTR procedure of Figure F.14.

Table F.2 – Extension to the left

i_0	$i_{left_{5\text{-}3}}$	$i_{left_{9\text{-}7}}$
even	1	3
odd	2	4

Table F.3 – Extension to the right

i_1	$i_{right_{5\text{-}3}}$	$i_{right_{9\text{-}7}}$
odd	1	3
even	2	4

F.3.8 The 1D_FILTR procedure

One reversible filtering procedure $1D\_FILTR_{5\text{-}3R}$ and one irreversible filtering procedure $1D\_FILTR_{9\text{-}7I}$ are specified, depending on whether the 5-3 reversible or 9-7 irreversible wavelet transformation is used.

As illustrated in Figure F.16, both procedures take as input an extended 1D signal Y_{ext}, the index of the first coefficient i_0, and the index of the coefficient i_1 immediately following the last coefficient ($i_1 - 1$). They both produce as output signal X.

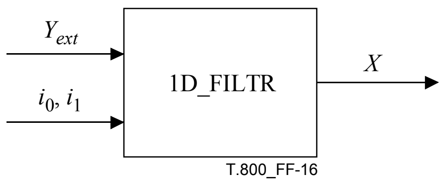


Figure F.16 – Parameters of the ID_FILTR procedure

Both procedures use lifting-based filtering, which consists in applying to the signal a sequence of very simple filtering operations called lifting steps, which alternately modify odd-indexed coefficient values of the signal with a weighted sum of even-indexed coefficient values, and even-indexed coefficient values with a weighted sum of odd-indexed coefficient values.

F.3.8.1 The 1D_FILTR$_{5\text{-}3R}$ procedure

The 1D_FILT5-3R procedure uses lifting-based filtering in conjunction with rounding operations. Equation (F-5) is first performed for all values of n indicated, followed by Equation (F-6) which uses values calculated from Equation (F-5):

$$X(2n) = Y_{ext}(2n) - \left\lfloor \frac{Y_{ext}(2n-1) + Y_{ext}(2n+1) + 2}{4} \right\rfloor \quad \text{for} \quad \left\lfloor \frac{i_0}{2} \right\rfloor \le n < \left\lfloor \frac{i_1}{2} \right\rfloor + 1 \qquad \text{(F-5)}$$

$$X(2n+1) = Y_{ext}(2n+1) + \left\lfloor \frac{X(2n) + X(2n+2)}{2} \right\rfloor \quad \text{for} \quad \left\lfloor \frac{i_0}{2} \right\rfloor \le n < \left\lfloor \frac{i_1}{2} \right\rfloor \qquad \text{(F-6)}$$

The values of $X(k)$ are such that $i_0 \le k < i_1$ form the output of the 1D_FILTR$_{5\text{-}3R}$ procedure.

F.3.8.2 The 1D_FILTR$_{9\text{-}7I}$ procedure

The 1D-FILTR$_{9\text{-}7I}$ procedure uses lifting-based filtering (there is no rounding operation). The lifting parameters (α, β, γ, δ) and the scaling parameter K for all filtering steps are defined in F.3.8.2.1.

Equation (F-7) describes the two scaling steps (1 and 2) and the four lifting steps (3 through 6) of the 1D filtering performed on the extended signal $Y_{ext}(n)$ to produce the $i_1 - i_0$ coefficients of signal X. These steps are performed in the following order.

Firstly, step 1 is performed for all values of n such that $\left\lfloor \frac{i_0}{2} \right\rfloor - 1 \le n < \left\lfloor \frac{i_1}{2} \right\rfloor + 2$, and step 2 is performed for all values of n such that $\left\lfloor \frac{i_0}{2} \right\rfloor - 2 \le n < \left\lfloor \frac{i_1}{2} \right\rfloor + 2$.

Then, step 3 is performed for all values of n such that $\left\lfloor \frac{i_0}{2} \right\rfloor - 1 \le n < \left\lfloor \frac{i_1}{2} \right\rfloor + 2$, and uses values calculated in steps 1 and 2.

Then, step 4 is performed for all values of n such that $\left\lfloor \frac{i_0}{2} \right\rfloor - 1 \le n < \left\lfloor \frac{i_1}{2} \right\rfloor + 1$, and uses values calculated in steps 2 and 3.

Then, step 5 is performed for all values of n such that $\left\lfloor \frac{i_0}{2} \right\rfloor \le n < \left\lfloor \frac{i_1}{2} \right\rfloor + 1$, and uses values calculated in steps 3 and 4.

Finally, step 6 is performed for all values of n such that $\left\lfloor \frac{i_0}{2} \right\rfloor \le n < \left\lfloor \frac{i_1}{2} \right\rfloor$, and uses values calculated in steps 4 and 5.

$$\begin{cases} X(2n) = KY_{ext}(2n) & [STEP1] \\ X(2n+1) = (1/K)Y_{ext}(2n+1) & [STEP2] \\ X(2n) = X(2n) - \delta(X(2n-1) + X(2n+1)) & [STEP3] \\ X(2n+1) = X(2n+1) - \gamma(X(2n) + X(2n+2)) & [STEP4] \\ X(2n) = X(2n) - \beta(X(2n-1) + X(2n+1)) & [STEP5] \\ X(2n+1) = X(2n+1) - \alpha(X(2n) + X(2n+2)) & [STEP6] \end{cases} \qquad \text{(F-7)}$$

where the values of the lifting parameters (α, β, γ, δ) and K are defined in Table F.4.

Table F.4 – Definition of lifting parameters for the 9-7 irreversible filter

Parameter	Exact expression	Approximate value
α	$-g_4 / g_3$	–1.586 134 342 059 924
β	g_3 / r_1	–0.052 980 118 572 961
γ	r_1 / s_0	0.882 911 075 530 934
δ	s_0 / t_0	0.443 506 852 043 971
K	$1 / t_0$	1.230 174 104 914 001

The values of $X(k)$ are such that $i_0 \le k < i_1$ form the output of the $1D\_FILTR_I$ procedure.

F.3.8.2.1 Filtering parameters for the $1D\_FILTR_{9\text{-}7I}$ procedure

The filtering parameters (α, β, γ, δ, K) are defined in Table F.4, in terms of parameters g_n from Table F.5, and parameters (r_0, r_1, s_0, t_0) from Table F.6. The parameters g_n are defined in terms of parameters x_1, $\Re x_2$ and $|x_2|^2$ given in Table F.7. All tables give a closed-form expression for all parameters, including approximations up to 15 decimal points.

Table F.5 – Definition of coefficients g_n

n	Coefficients g_n	Approximate value of g_n
0	$5x_1\left(48\lvert x_2\rvert^2 - 16\Re x_2 + 3\right)/32$	–0.602 949 018 236 360
1	$-5x_1\left(8\lvert x_2\rvert^2 - \Re x_2\right)/8$	0.266 864 118 442 875
2	$5x_1\left(4\lvert x_2\rvert^2 + 4\Re x_2 - 1\right)/16$	0.078 223 266 528 990
3	$-5x_1(\Re x_2)/8$	–0.016 864 118 442 875
4	$5x_1 / 64$	–0.026 748 757 410 810

Table F.6 – Intermediate expressions (r_0, r_1, s_0, t_0)

Parameter	Exact expression	Approximate value
r_0	$-g_0 + 2g_1g_4 / g_3$	1.449 513 704 087 943
r_1	$-g_2 + g_4 + g_1g_4 / g_3$	0.318 310 318 985 991
s_0	$g_1 - g_3 - g_3r_0 / r_1$	0.360 523 644 801 462
t_0	$r_0 - 2r_1$	0.812 893 066 115 961

Table F.7 – Intermediate expressions

Parameter	Exact expression	Approximate value
A	$\sqrt[3]{\frac{63-14\sqrt{15}}{1080\sqrt{15}}}$	0.128 030 244 703 494
B	$-\sqrt[3]{\frac{63+14\sqrt{15}}{1080\sqrt{15}}}$	–0.303 747 672 895 197
x_1	A + B – 1 / 6	–0.342 384 094 858 369
$\Re x_2$	$-\frac{(A+B)}{2}-\frac{1}{6}$	–0.078 807 952 570 815
$\lvert x_2 \rvert^2$	$\left[\frac{(A+B)}{2}+\frac{1}{6}\right]^2+\frac{3(A-B)^2}{4}$	0.146 034 820 982 800

F.4 Forward transformation (informative)

F.4.1 The FDWT procedure (informative)

The forward discrete wavelet transformation (FDWT) transforms DC-level shifted tile-component samples $I(x, y)$ into a set of sub-bands with coefficients $a_b(u_b, v_b)$ (FDWT procedure). The FDWT procedure (see Figure F.17) also takes as input the number of decomposition levels N_L signalled in the COD or COC markers (see A.6.1 and A.6.2).

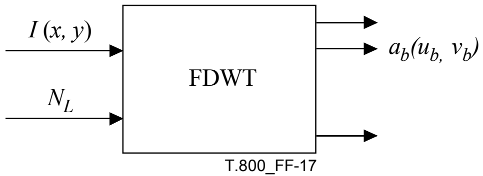


Figure F.17 – Inputs and outputs of the FDWT procedure

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

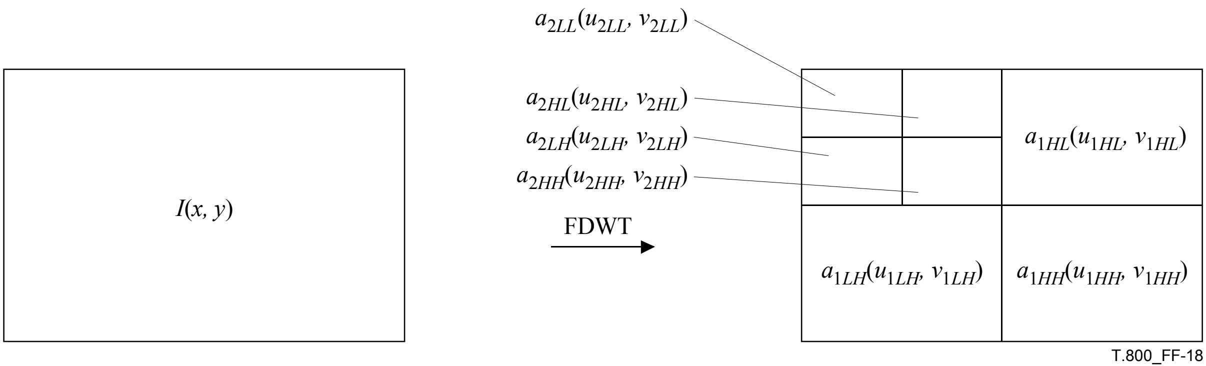


Figure F.18 – The FDWT ($N_L = 2$)

The FDWT procedure starts with the initialization of the variable *lev* (the current decomposition level) to zero, and setting the sub-band $a_{0LL}(u_{0LL}, v_{0LL})$ to the input array $I(u, v)$. The 2D_SD procedure is performed at every level *lev*, where the level *lev* increases by one at each iteration, and until N_L iterations are performed. The 2D_SD procedure is iterated over the *lev*LL sub-band produced at each iteration.

As defined in Annex B (see Equation (B-15)), the coordinates of the sub-band $a_{levLL}(u, v)$ lie in the range defined by:

$$tbx_0 \le u < tbx_1 \quad and \quad tby_0 \le v < tby_1 \tag{F-8}$$

Figure F.19 describes the FDWT procedure.

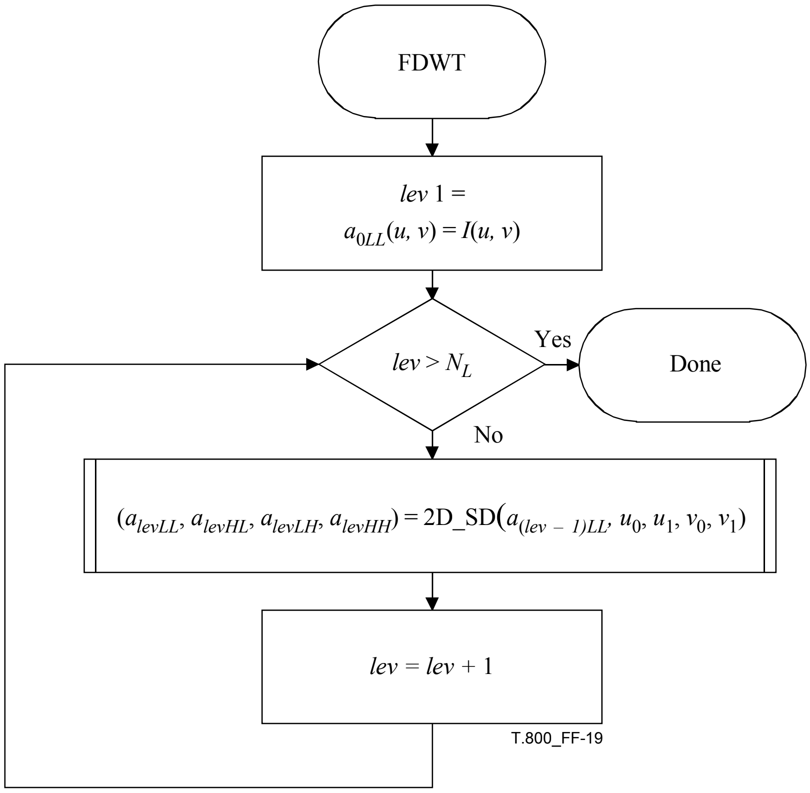


Figure F.19 – The FDWT procedure

F.4.2 The 2D_SD procedure (informative)

The 2D_SD procedure performs a decomposition of a two-dimensional array of coefficients or samples $a_{(lev-1)LL}(u, v)$ into four groups of sub-band coefficients $a_{levLL}(u, v)$, $a_{levHL}(u, v)$, $a_{levLH}(u, v)$, and $a_{levHH}(u, v)$.

The total number of coefficients of the lev_{LL} sub-band is equal to the sum of the total number of coefficients of the four sub-bands resulting from the 2D_SD procedure.

Figure F.20 describes the input and output parameters of the 2D_SD procedure.

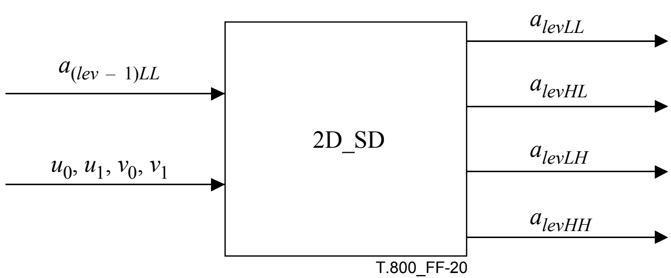


Figure F.20 – Inputs and outputs of the 2D_SD procedure

Figure F.21 illustrates the sub-band decomposition performed by the 2D_SD procedure.

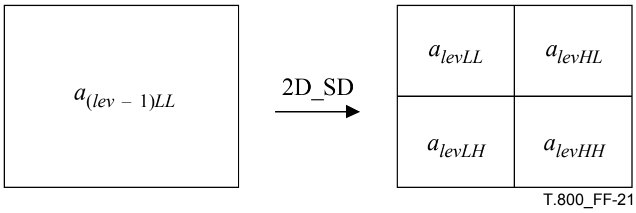


Figure F.21 – One-level decomposition into four sub-bands (2D_SD procedure)

The 2D_SD procedure first applies the VER_SD procedure to all columns of $a(u, v)$. It then applies the HOR_SD procedure to all rows of $a(u, v)$. The coefficients thus obtained from $a(u, v)$ are deinterleaved into the four sub-bands using the 2D_DEINTERLEAVE procedure.

Figure F.22 describes the 2D_SD procedure.

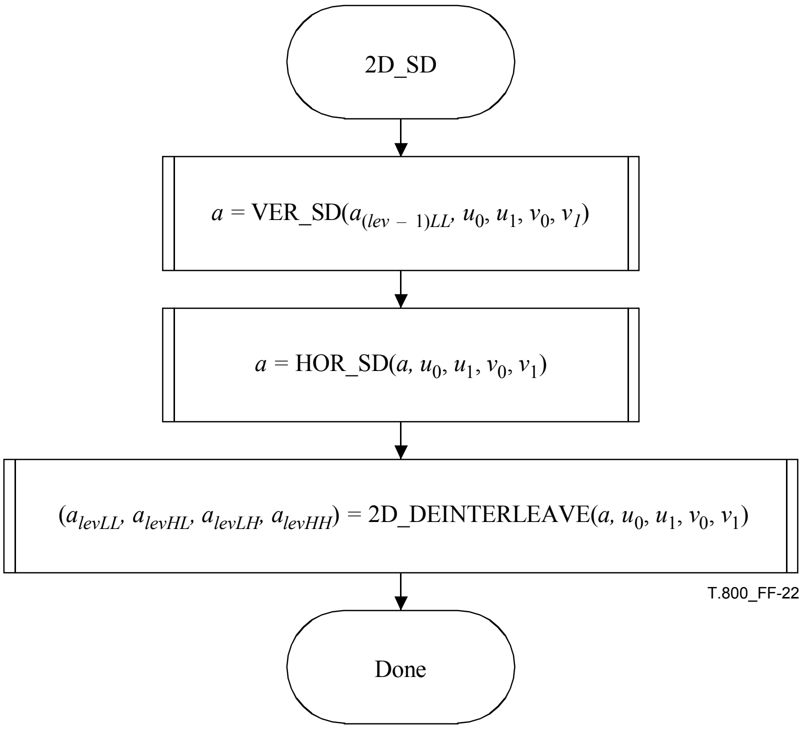


Figure F.22 – The 2D_SD procedure

F.4.3 The VER_SD procedure (informative)

The VER_SD procedure performs a vertical sub-band decomposition of a two-dimensional array of coefficients. It takes as input the two-dimensional array $a_{(lev-1)LL}(u, v)$, the horizontal and vertical extent of its coefficients as indicated by $u_0 \le u < u_1$ and $v_0 \le v < v_1$ (see Figure F.23) and produces as output a vertically filtered version $a(u, v)$ of the input array, column by column. The values of u_0, u_1, v_0, v_1 used by the VER_SD procedure are those of tbx_0, tbx_1, tby_0, tby_1 corresponding to sub-band $b = (lev - 1)$LL (see definition in Equation (B-15)).

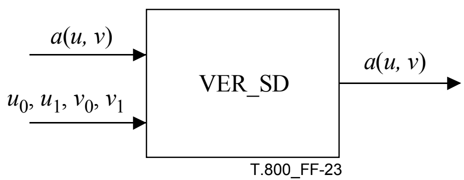


Figure F.23 – Inputs and outputs of the VER_SD procedure

As illustrated in Figure F.24, the VER_SD procedure applies the one-dimensional sub-band decomposition (1D_SD procedure) to each column of the input array $a(u, v)$, and stores the result back into each column.

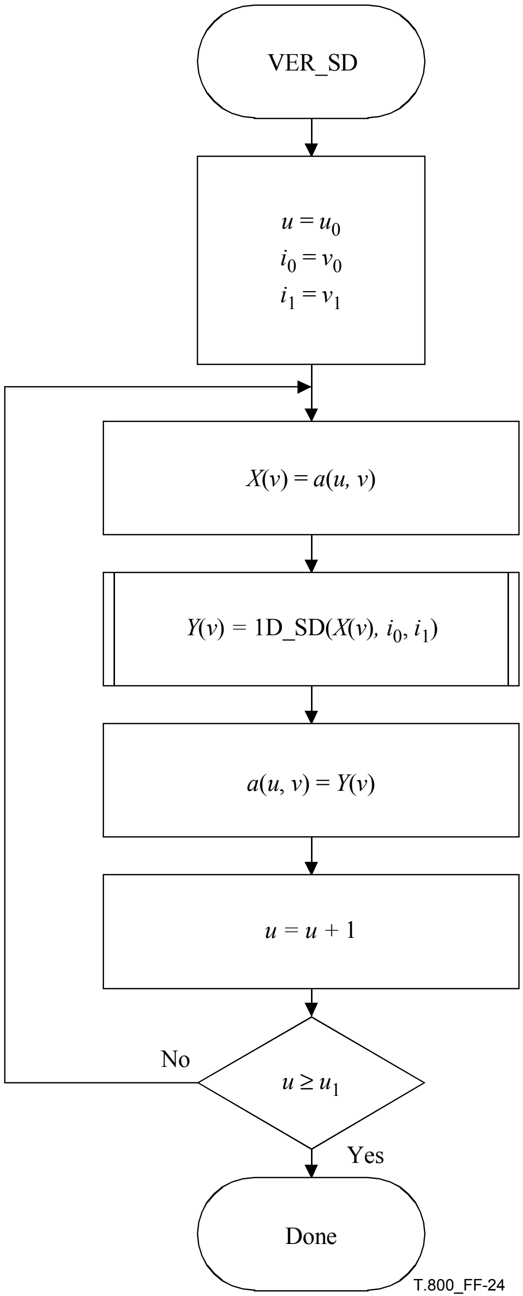


Figure F.24 – The VER_SD procedure

F.4.4 The HOR_SD procedure (informative)

The HOR_SD procedure performs a horizontal sub-band decomposition of a two-dimensional array of coefficients. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical extent of its coefficients as indicated by $u_0 \leq u < u_1$ and $v_0 \leq v < v_1$ (see Figure F.25) and produces as output a horizontally filtered version of the input array, row by row.

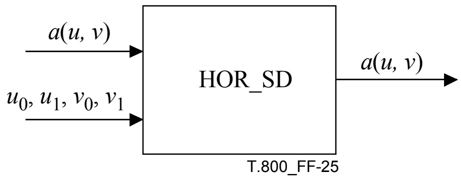


Figure F.25 – Inputs and outputs of the HOR_SD procedure

As illustrated in Figure F.26, the HOR_SD procedure applies the one-dimensional sub-band decomposition (1D_SD procedure) to each row of the input array $a(u, v)$ and stores the result back in each row.

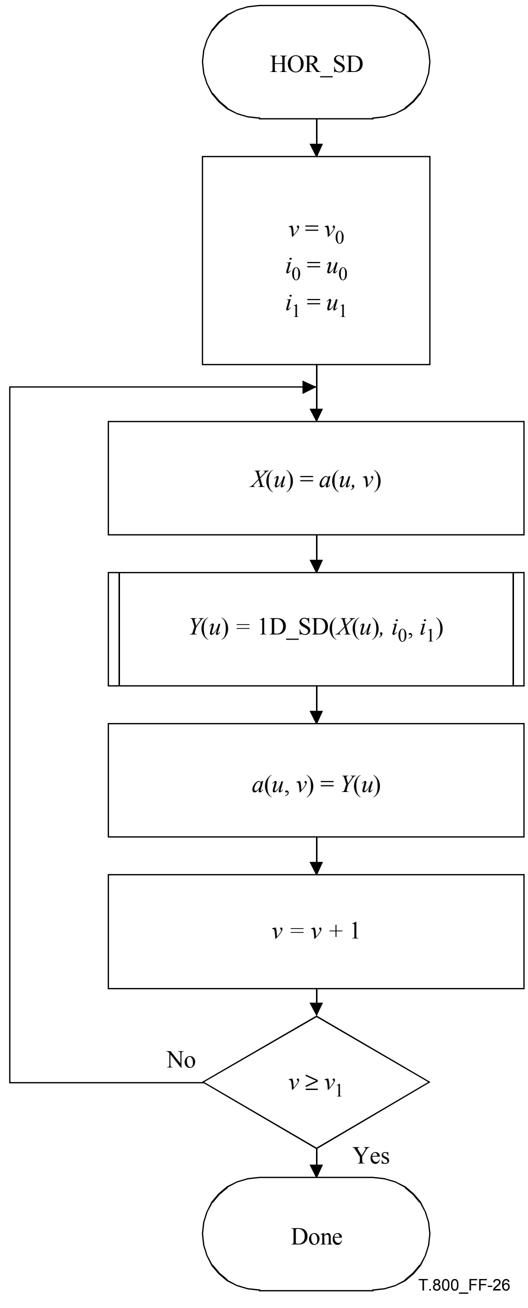


Figure F.26 – The HOR_SD procedure

F.4.5 The 2D_DEINTERLEAVE procedure (informative)

As illustrated in Figure F.27, the 2D_DEINTERLEAVE procedure deinterleaves the coefficients of $a(u, v)$ into four sub-bands. The arrangement is dependent on the coordinates (u_0, v_0) of the first coefficient of $a(u, v)$.

The way these sub-bands are formed from the output $a(u, v)$ of the HOR_SD procedure is described by the 2D_DEINTERLEAVE procedure illustrated in Figure F.28.

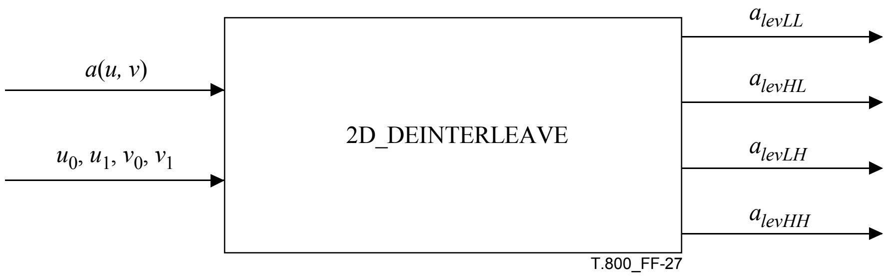


Figure F.27 – Parameters of 2D_DEINTERLEAVE procedure

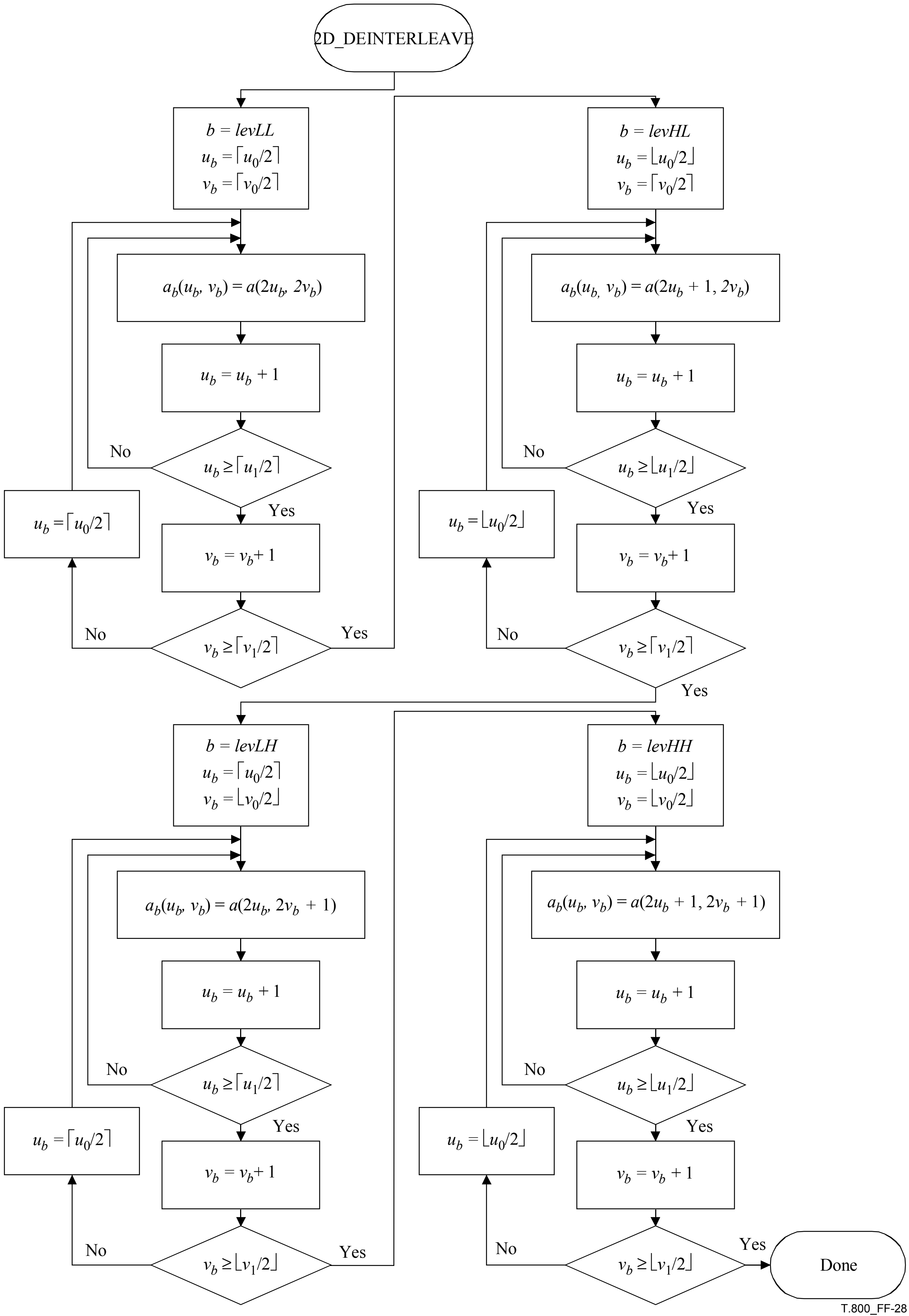


Figure F.28 – The 2D_DEINTERLEAVE procedure

F.4.6 The 1D_SD procedure (informative)

As illustrated in Figure F.29, the 1D_SD procedure takes as input a one-dimensional array *X(i)*, the extent of its coefficients as indicated by $i_0 \le i < i_1$. It produces as output an array *Y(i)*, with the same indices (i_0, i_1).

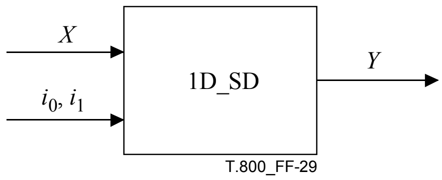


Figure F.29 – Parameters of the 1D_SD procedure

For signals of length one (i.e., $i_0 = i_1 - 1$), the 1D_SD procedure sets the value of $Y(i_0)$ to $Y(i_0) = X(i_0)$ if i_0 is an even integer, and to $Y(i_0) = 2X(i_0)$ if i_0 is an odd integer.

For signals of length greater than or equal to two (i.e., $i_0 < i_1 - 1$), as illustrated in Figure F.30, the 1D_SD procedure first uses the 1D_EXTD procedure to extend the signal *X* beyond its left and right boundaries resulting in the extended signal X_{ext}, and then uses the 1D_FILTD procedure to filter the extended signal X_{ext} and produce the desired filtered signal *Y*.

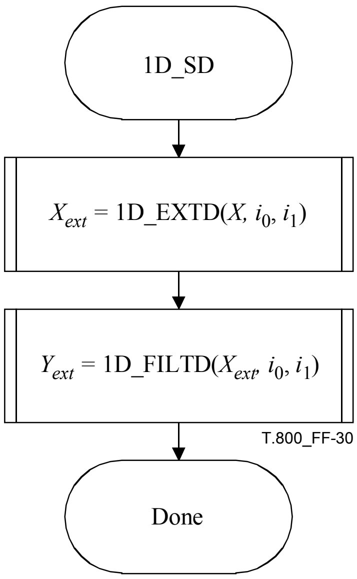


Figure F.30 – The 1D_SD procedure

F.4.7 The 1D_EXTD procedure (informative)

The 1D_EXTD procedure is identical to the 1D_EXTR procedure, except for the values of the $i_{left_{9\text{-}7}}$, $i_{right_{9\text{-}7}}$, $i_{left_{5\text{-}3}}$ and $i_{right_{5\text{-}3}}$ parameters, which are given in Tables F.8 and F.9.

Table F.8 – Extension to the left

i_0	$i_{left_{5\text{-}3}}$	$i_{left_{9\text{-}7}}$
even	2	4
odd	1	3

Table F.9 – Extension to the right

i_1	$i_{right_{5\text{-}3}}$	$i_{right_{9\text{-}7}}$
odd	2	4
even	1	3

F.4.8 The 1D_FILTD procedure (informative)

This Recommendation | International Standard specifies one irreversible procedure ($1D\_FILTD_{9\text{-}7I}$) and one reversible filtering procedure ($1D\_FILTD_{5\text{-}3R}$), depending on whether the 9-7 irreversible or 5-3 reversible wavelet transformation is selected.

As illustrated in Figure F.31, both procedures take as input an extended 1D signal X_{ext}, the index of the first coefficient i_0, and the index of the coefficient i_1 immediately following the last coefficient ($i_1 - 1$). They both produce an output signal, Y. The even-indexed coefficients of the Y signal are a low-pass downsampled version of the extended signal X_{ext}, while the odd-indexed coefficients of the signal Y are a high-pass downsampled version of the extended signal X_{ext}.

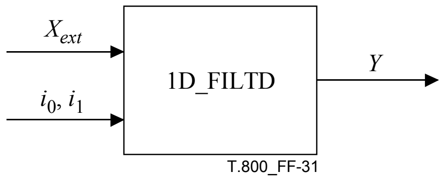


Figure F.31 – Parameters of the 1D_FILTD procedure

F.4.8.1 The $1D\_FILTD_{5\text{-}3R}$ procedure (informative)

The reversible transformation described in this clause is the reversible lifting-based implementation of filtering by the 5-3 reversible wavelet filter. The reversible transformation is defined using lifting-based filtering. The odd-indexed coefficients of output signal Y are computed first for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil - 1 \le n < \left\lceil \frac{i_1}{2} \right\rceil$ as given in Equation (F-9):

$$Y(2n+1) = X_{ext}(2n+1) - \left\lfloor \frac{X_{ext}(2n) + X_{ext}(2n+2)}{4} \right\rfloor \tag{F-9}$$

Then the even-indexed coefficients of output signal Y are computed from the even-indexed values of extended signal X_{ext} and the odd-indexed coefficients of signal Y for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil \le n < \left\lceil \frac{i_1}{2} \right\rceil$ as given in Equation (F-10):

$$Y(2n) = X_{ext}(2n) + \left\lfloor \frac{Y(2n-1) + Y(2n+1) + 2}{4} \right\rfloor \tag{F-10}$$

The values of $Y(k)$ such that $i_0 \le k\ i_1$ form the output of the $1D\_FILTD_R$ procedure.

F.4.8.2 The 1D_FILTDI procedure (informative)

The irreversible transformation described in this clause is the lifting-based DWT implementation of filtering by the 9-7 irreversible filter.

Equation (F-11) describes the four lifting steps (1 through 4) and the two scaling steps (5 and 6) of the 1D filtering performed on the extended signal $X_{ext}(n)$ to produce the $i_1 - i_0$ coefficients of signal Y. These steps are performed in the following order.

Firstly, step 1 is performed for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil - 2 \le n < \left\lceil \frac{i_1}{2} \right\rceil + 1$.

Then, step 2 is performed for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil - 1 \le n < \left\lceil \frac{i_1}{2} \right\rceil + 1$, and uses values calculated at step 1.

Then, step 3 is performed for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil - 1 \le n < \left\lceil \frac{i_1}{2} \right\rceil$, and uses values calculated at steps 1 and 2.

Then, step 4 is performed for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil \le n < \left\lceil \frac{i_1}{2} \right\rceil$, and uses values calculated at steps 2 and 3.

Finally, step 5 is performed for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil \le n < \left\lceil \frac{i_1}{2} \right\rceil$ and uses values calculated at step 3, and step 6 is performed for all values of n such that $\left\lceil \frac{i_0}{2} \right\rceil \le n < \left\lceil \frac{i_1}{2} \right\rceil$ and uses values calculated at step 4.

$$\begin{cases} Y(2n+1) = X_{ext}(2n+1) + \alpha\left(X_{ext}(2n) + X_{ext}(2n+2)\right) & [STEP1] \\ Y(2n) = X_{ext}(2n) + \beta\left(Y(2n-1) + Y(2n+1)\right) & [STEP2] \\ Y(2n+1) = Y(2n+1) + \gamma\left(Y(2n) + Y(2n+2)\right) & [STEP3] \\ Y(2n) = Y(2n) + \delta\left(Y(2n-1) + Y(2n+1)\right) & [STEP4] \\ Y(2n+1) = KY(2n+1) & [STEP5] \\ Y(2n) = (1/K)\,Y(2n) & [STEP6] \end{cases} \quad \text{(F-11)}$$

where the values of the lifting parameters α, β, γ, δ, and K are defined in Table F.4.

The values of such that $i_0 \le k < i_1$ form the output of the 1D_FILTD$_I$ procedure.

Annex G

DC level shifting and multiple component transformations

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

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

This annex specifies DC level shifting that converts the signed values resulting from the decoding process to the proper reconstructed samples.

This annex also describes two different multiple component transformations. These multiple component transformations are used to improve compression efficiency. They are not related to multiple component transformations used to map colour values for display purposes. One multiple component transformation is reversible and may be used for lossy or lossless coding. The other is irreversible and may only be used for lossy coding.

G.1 DC level shifting of tile-components

Figure G.1 shows the flow of DC level shifting in the system with a multiple component transformation.

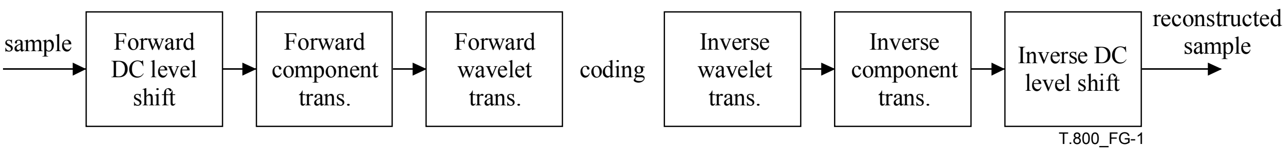


Figure G.1 – Placement of the DC level shifting with component transformation

Figure G.2 shows the flow of DC level shifting in the system without a multiple component transformation.

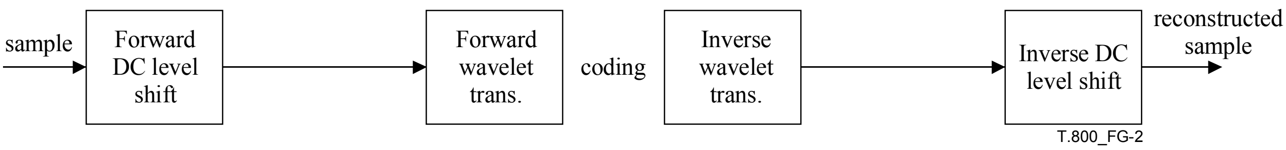


Figure G.2 – Placement of the DC level shifting without component transformation

G.1.1 DC level shifting of tile-components (informative)

DC level shifting is performed on samples of components that are unsigned only. It is performed prior to computation of a forward multiple component transformation (RCT or ICT), if one is used. Otherwise it is performed prior to the wavelet transformation described in Annex F. If the MSB of $Ssiz^i$ from the SIZ marker segment (see A.5.1) is zero, all samples $I(x, y)$ of the ith component are level shifted by subtracting the same quantity from each sample as follows:

$$I(x,y) \leftarrow I(x,y) - 2^{Ssiz^i} \quad \text{(G-1)}$$

G.1.2 Inverse DC level shifting of tile-components

Inverse DC level shifting is performed on reconstructed samples of components that are unsigned only. It is performed after to computation of the inverse multiple component transformation (RCT or ICT), if one is used. Otherwise it is performed after the inverse wavelet transformation described in Annex F. If the MSB of $Ssiz^i$ from the SIZ marker segment (see A.6.1) is zero, all samples $I(x, y)$ of the ith component are level shifted by adding the same quantity from each sample as follows:

$$I(x,y) \leftarrow I(x,y) + 2^{Ssiz^i} \quad \text{(G-2)}$$

NOTE – Due to quantization effects, the reconstructed samples *I(x, y)* may exceed the dynamic range of the original samples. There is no normative procedure for this overflow or underflow situation. However, clipping the value to the nearest value within the original dynamic range is a typical solution.

G.2 Reversible multiple component transformation (RCT)

The use of the reversible multiple component transformation is signaled in the COD marker segment (see A.6.1). The RCT shall be used only with the 5-3 reversible filter. The RCT is a decorrelating transformation applied to the first three components of an image (indexed as 0, 1 and 2). The three components input into the RCT shall have the same separation on the reference grid and the same bit-depth.

NOTE – While the RCT is reversible, and thus capable of lossless compression, it may be used in truncated codestreams to provide lossy compression.

G.2.1 Forward RCT (informative)

Prior to applying the Forward RCT, the image component samples are DC level shifted, for unsigned components.

The Forward RCT is applied to components $I_0(x, y)$, $I_1(x, y)$, $I_2(x, y)$ as follows:

$$Y_0(x,y) = \left\lfloor \frac{I_0(x,y) + 2I_1(x,y) + I_2(x,y)}{4} \right\rfloor \tag{G-3}$$

$$Y_1(x,y) = I_2(x,y) - I_1(x,y) \tag{G-4}$$

$$Y_2(x,y) = I_0(x,y) - I_1(x,y) \tag{G-5}$$

If I_0, I_1, and I_2 are normalized to the same precision, then Equations (G-4) and (G-5) result in a numeric precision of Y_1 and Y_2 that is one bit greater than the precision of the original components. This increase in precision is necessary to ensure reversibility.

G.2.2 Inverse RCT

After the inverse wavelet transformation is preformed as described in Annex F, the following Inverse RCT is applied:

$$I_1(x,y) = Y_0(x,y) - \left\lfloor \frac{Y_2(x,y) + Y_1(x,y)}{4} \right\rfloor \tag{G-6}$$

$$I_0(x,y) = Y_2(x,y) + I_1(x,y) \tag{G-7}$$

$$I_2(x,y) = Y_1(x,y) + I_1(x,y) \tag{G-8}$$

After applying the Inverse RCT, the unsigned image components are inverse DC level shifted.

G.3 Irreversible multiple component transformation (ICT)

This clause specifies an irreversible multiple component transformation. The use of the irreversible component transformation is signaled in the COD marker segment (see A.6.1). The ICT shall be used only with the 9-7 irreversible filter. The ICT is a decorrelating transformation applied to the first three components of an image (indexed as 0, 1 and 2). The three components input into the ICT shall have the same separation on the reference grid and the same bit-depth.

G.3.1 Forward ICT (informative)

The Forward ICT is applied to image component samples $I_0(x, y)$, $I_1(x, y)$, $I_2(x, y)$, as follows:

$$Y_0(x,y) = -\,0{,}299\; I_0(x,y) - 0{,}587\; I_1(x,y) + 0{,}114\; I_2(x,y) \tag{G-9}$$

$$Y_1(x,y) = -0{,}16875\ \ I_0(x,y) - 0{,}331260\ \ I_1(x,y) + 0{,}5\ \ I_2(x,y) \qquad \text{(G-10)}$$

$$Y_2(x,y) = 0{,}5\ \ I_0(x,y) - 0{,}41869\ \ I_1(x,y) - 0{,}08131\ \ I_2(x,y) \qquad \text{(G-11)}$$

NOTE – If the first three components are Red, Green and Blue components, then the Forward ICT is of a YCbCr transformation.

G.3.2 Inverse ICT

After inverse wavelet transformation is performed as described in Annex F, the following Inverse ICT is applied:

$$I_0(x,y) = Y_0(x,y) + 1{,}402\ \ Y_2(x,y) \qquad \text{(G-12)}$$

$$I_1(x,y) = Y_0(x,y) - 0{,}34413\ \ Y_1(x,y) - 0{,}71414\ \ Y_2(x,y) \qquad \text{(G-13)}$$

$$I_2(x,y) = Y_0(x,y) + 1{,}772\ \ Y_1(x,y) \qquad \text{(G-14)}$$

Equations (G-12), (G-13) and (G-14) do not imply a required precision for the coefficients. After applying the Inverse ICT, the unsigned image component samples are inverse DC level shifted.

G.4 Chrominance component sub-sampling and the reference grid

The relationship between the components and the reference grid is signalled in the SIZ marker (see A.5.1) and described in B.2.

Annex H

Coding of images with regions of interest

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

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

This annex describes the region of interest (ROI) technology. An ROI is a part of an image that is coded earlier in the codestream than the rest of the image (the background). The coding 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 Decoding of ROI

The procedure specified in this clause is applied only in the case of the presence of an RGN marker segment (indicating the presence of an ROI).

The procedure realigns the significant bits of ROI coefficients and background coefficients. It is defined using the following steps:

1) Get the scaling value, *s*, from the SPrgn parameter of the RGN marker segment in the codestream (see A.6.3). The following steps (2, 3 and 4) are applied to each coefficient (*u*, *v*) of sub-band *b*.
2) If $N_b(u, v) < M_b$ (see definition of $N_b(u, v)$ in D.2.1 and of M_b in Equation (E-2)), then no modification takes place.
3) If $N_b(u, v) \geq M_b$ and if at least one of the first M_b (see definition in E.1) MSBs ($i = 1, ..., M_b$) is non-zero, then the value of $N_b(u, v)$ is updated as $N_b(u, v) = M_b$.
4) If $N_b(u, v) \geq M_b$ and if all first M_b MSBs are equal to zero, then the following modifications are made:
   a) discard the first *s* MSBs and shift the remaining MSBs *s* places, as described in Equation (H-1), for $i = 1, ..., M_b$:

$$MSB_i(b,u,v) = \begin{cases} MSB_{i+s}(b,u,v) & \text{if } i+s \leq N_b(u,v) \\ 0 & \text{if } i+s > N_b(u,v) \end{cases} \quad \text{(H-1)}$$

   b) update the value of $N_b(u, v)$ as given in Equation (H-2):

$$N_b(u,v) = \max\left(0, N_b(u,v) - s\right) \quad \text{(H-2)}$$

H.2 Description of the Maxshift method

H.2.1 Encoding of ROI (informative)

The encoding of the quantized transform coefficients is done in a similar way to encoding without any ROIs. At the encoder side an 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 lossless). The ROI mask is a bit map describing these coefficients. See H.3 for details on how the mask is generated.

The quantized transform coefficients outside of the ROI mask, 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 method can be described using the following steps:

1) Generate ROI mask, *M(x, y)* (see H.3).
2) Find the scaling value *s* (see H.2.2).
3) Add *s* LSBs to each coefficient $\left|q_b(u,v)\right|$. The number M'_b of magnitude bit-planes will then be:

$$M'_b = M_b + s \quad \text{(H-3)}$$

where M_b is given by Equation (E-2) and the new value of each coefficient is given by:

$$|q_b(u,v)| = |q_b(u,v)| \cdot 2^s \quad \text{(H-4)}$$

4) Scale down all background coefficients given by $M(x, y)$ using the scaling value s (see H.3). Thus, if $|q_b(u,v)|$ is a background coefficient given by $M(x, y)$, then:

$$|q_b(u,v)| = \frac{|q_b(u,v)|}{2^s} \quad \text{(H-5)}$$

5) Write the scaling value s into the codestream using the SPrgn parameter of the RGN marker segment.

After these steps the quantized transform coefficients are entropy coded as usual.

H.2.2 Selection of scaling value, s, at encoder side (informative)

The scaling value, s, may be chosen so that Equation (H-6) holds, 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.

$$s \geq max(M_b) \quad \text{(H-6)}$$

This guarantees that the scaling value used will be sufficiently large to ensure all the significant bits associated with the ROI will be in higher bit-planes than all the significant bits associated with the background.

H.3 Remarks on region of interest coding (informative)

The ROI functionality described in H.2 depends only on the scaling value chosen on the encoder side and hence only on the amplitude of the coefficients on the decoder side. It is up to the encoder to generate a mask that corresponds to the coefficients that need to be encoded with better quality to yield an ROI with better quality than the background. Clause H.3.1 describes how to generate the ROI mask for a particular region in the image. Clause H.3.2 describes how to generate the mask in the case of multi-component images and H.3.3 describes how to generate the ROI mask for disjoint regions. Clause H.3.4 describes a possible way to deal with the increase of coefficient bit depth. Clause H.3.5 describes how the ROI mask can be extended so as to not correspond exactly to a region in the image domain and how the Maxshift method may be used to encode the ROI and the background with different quality.

H.3.1 Region of interest mask generation (informative)

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).

To illustrate the concept of ROI mask generation, let us restrict ourselves to a single ROI and a single image component, and identify the samples that belong to the ROI in the image domain by a binary mask, $M(x, y)$, where:

$$M(x,y) = \begin{cases} 1 & \text{wavelet coefficient } (x,y) \text{ is needed} \\ 0 & \text{accuracy on } (x,y) \text{ can be sacrificed without affecting ROI} \end{cases} \quad \text{(H-7)}$$

The mask is a map of the ROI in the wavelet domain so that it has a non-zero value inside the ROI and 0 outside. In each step the LL sub-band of the mask is then updated row by row and then column by column. The mask will then indicate which coefficients are needed at this step so that the inverse wavelet transformation will reproduce the coefficients of the previous mask.

For example, the last step of the inverse wavelet transformation is a composition of two sub-bands into one. Then to trace this step backwards, one finds the coefficients of both sub-bands that are needed. 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 wavelet transformation 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 losslessly).

Given below is a description of how the expansion of the mask is acquired from the various filters. Similar methods can be used for other filters.

H.3.1.1 Region of interest mask generation using the 5-3 reversible filter (informative)

In order to get the optimal set of quantized coefficients to be scaled, the following equations described in this clause should be used.

To see what coefficients need to be in the mask, the inverse wavelet transformation is studied. Equations (F-5) and (F-6) give the coefficients needed to reconstruct $X(2n)$ and $X(2n+1)$ losslessly. It can immediately be seen that these are $L(n)$, $L(n+1)$, $H(n-1)$, $H(n)$, $H(n+1)$ (see Figure H.1). Hence if $X(2n)$ and $X(2n+1)$ are in the ROI, the listed low and high sub-band coefficients are in the mask. Notice that $X(2n)$ and $X(2n+1)$ are even and odd indexed points respectively, relative to the origin of the reference grid.

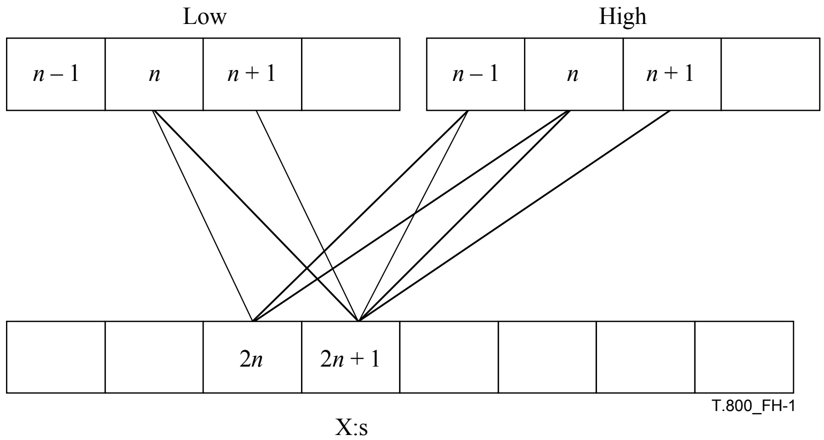


Figure H.1 – The inverse wavelet transformation with the 5-3 reversible filter

H.3.1.2 Region of interest mask generation using the 9-7 irreversible 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 clause should be used.

To see what coefficients need to be in the mask, the inverse wavelet transformation is studied as in H.3.1.1. Figure H.2 shows this $X(2n)$ and $X(2n+1)$ are even and odd indexed points respectively, related to the origin of the reference grid.

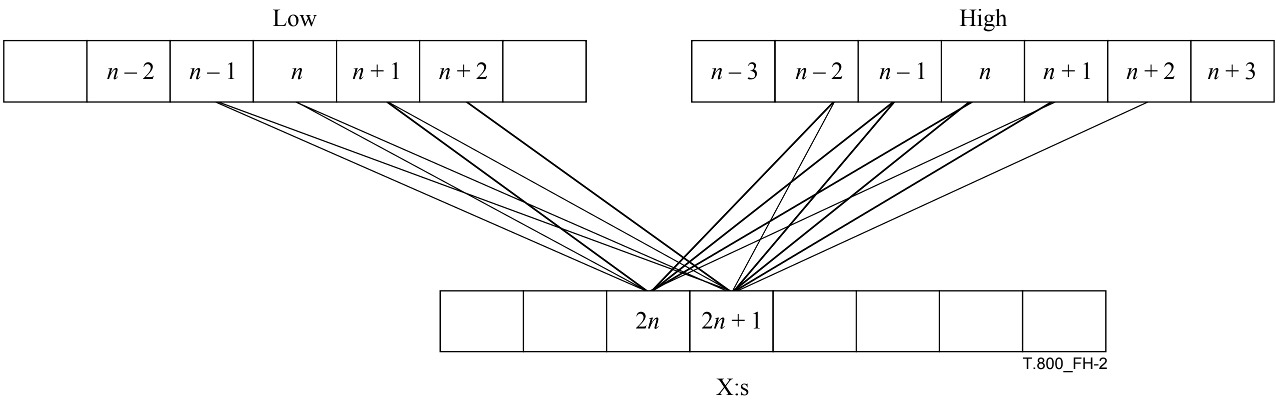


Figure H.2 – The inverse wavelet transformation with the 9-7 irreversible filter

The coefficients needed to reconstruct $X(2n)$ and $X(2n + 1)$ losslessly can immediately be seen to be $L(n - 1)$ to $L(n + 2)$ and $H(n - 2)$ to $H(n + 2)$. Hence if $X(2n)$ and $X(2n + 1)$ are in the ROI, those Low and High sub-band coefficients are in the mask.

H.3.2 Multi-component remark (informative)

For the case of colour images, the method applies separately in each colour component. If some of the colour components are down-sampled, the mask for the down-sampled components is created in the same way as the mask for the non-down-sampled components.

H.3.3 Disjoint regions remark (informative)

If the ROI consists of disjoint parts, then all parts have the same scaling value s.

H.3.4 Implementation precision remark (informative)

This ROI coding method might in some cases create situations where the dynamic range is exceeded. This is however easily solved by simply discarding the least significant bit-planes that exceed the limit due to the down-scaling operation. The effect will be that the ROI will have better quality than the background, even though the entire bit stream is decoded. It might however create problems when the image is coded with ROIs in a lossless mode. Discarding least significant bit-planes for the background might result in the background not being coded losslessly and in the worst case not being reconstructed at all. This depends on the dynamic range available.

H.3.5 An example of the usage of the Maxshift method (informative)

The Maxshift method, as described above, allows the user/application to specify multiple regions of arbitrary shape, which will be assigned higher priority compared to the rest of the image. The method does not require encoding or decoding of the ROI shape.

The Maxshift method allows the implementers of an encoder to exploit a number of functionalities that are supported by a compliant decoder. For example, it is possible to use the Maxshift method to encode an image with different quality for the ROI and the background. The image is quantized so that the ROI gets the desired quality (lossy or lossless) and then the Maxshift method is applied. If the image is encoded progressively by layer, not all of the layers of the wavelet coefficients belonging to the background need be encoded. This corresponds to using different quantization steps for the ROI and the background.

If the ROI is to be encoded losslessly, the most optimal set of wavelet coefficients giving a lossless result for the ROI is described by the mask generated using the algorithms described in H.3.1 However, the Maxshift method supports the use of any mask since the decoder does not need to generate the mask. Thus, it is possible for the encoder to include an entire sub-band, e.g. the low-low sub-band, in the ROI mask and thus send a low-resolution version of the background at an early stage of the progressive transmission. This is done by scaling all the quantized transform coefficients of the entire sub-band. In other words, the user can decide in which sub-band he will start coding ROI and thus, it is not necessary to wait for the entire ROI before receiving any information for the background.

Annex I

JP2 file format syntax

(This annex forms an integral part of this Recommendation | International Standard.
This annex is optional for the minimum decoder.)

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

I.1 File format scope

This annex defines an optional file format that applications may choose to use to wrap JPEG 2000 compressed image data. While not all applications will use this format, many applications will find that this format meets their needs. However, those applications that do implement this file format shall implement it as described in this entire annex.

This annex:

- specifies a binary container for both image and metadata;
- specifies a mechanism to indicate image properties, such as the tonescale or colourspace 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.

I.2 Introduction to the JP2 file format

The JPEG 2000 file format (JP2 file format) provides a foundation for storing application specific data (metadata) in association with a JPEG 2000 codestream, such as 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 file format encapsulates the JPEG 2000 codestream along with other core pieces of information about that codestream. The building-block of the JP2 file format is called a box. All information contained within the JP2 file is encapsulated in boxes. This Recommendation | International Standard defines several types of boxes; the definition of each specific box type defines the kinds of information that may be found within a box of that type. Some boxes will be defined to contain other boxes.

I.2.1 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 should allow mixed case for the alphabetic characters). On Macintosh file systems, JP2 files should be given the type code 'jp2\040'.

I.2.2 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 I.4.

Logically, the structure of a JP2 file is as shown in Figure I.1. 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.

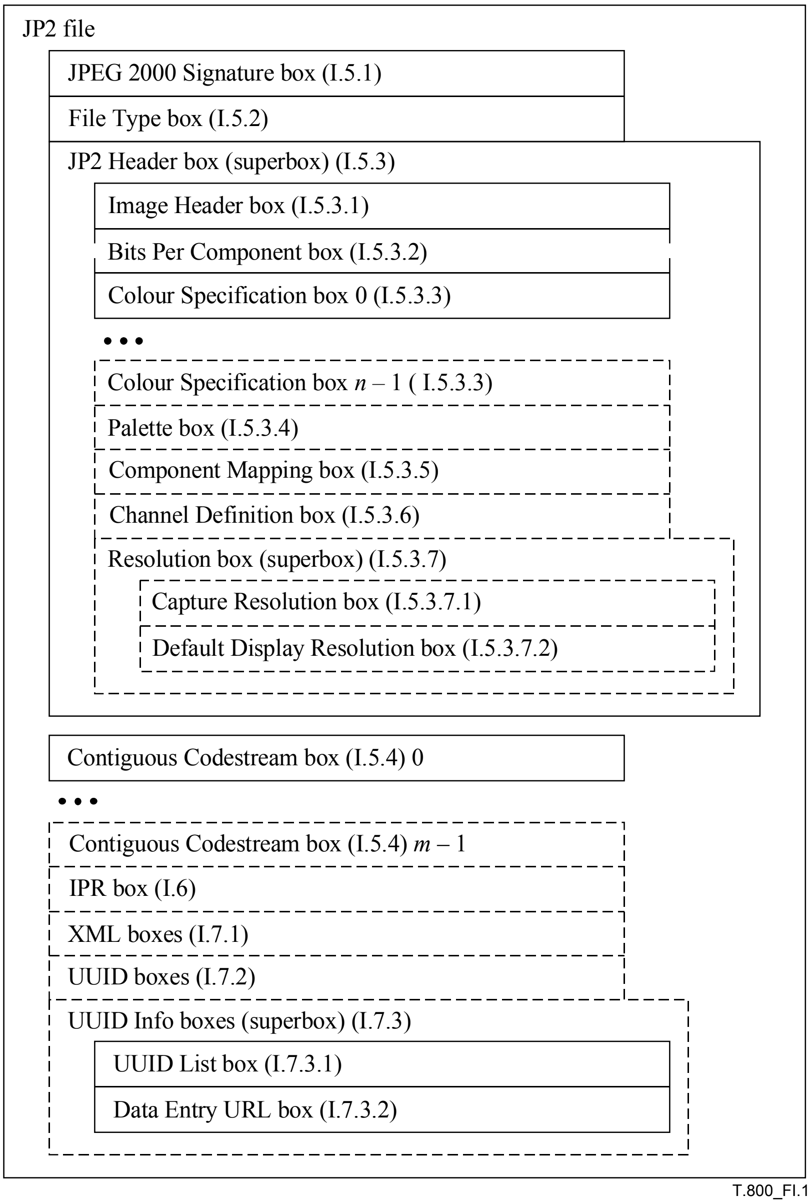


Figure I.1 – Conceptual structure of a JP2 file

Figure I.1 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 JPEG 2000 Signature box shall be the first box in a JP2 file, the File Type box shall immediately follow the JPEG 2000 Signature box and the JP2 Header box shall fall before the Contiguous Codestream box.

The file shown in Figure I.1 is a strict sequence of boxes. Other boxes may be found between the boxes defined in this Recommendation | International Standard. However, all information contained within a JP2 file shall be in the box format; byte-streams not in the box format shall not be found in the file.

As shown in Figure I.1, a JP2 file contains a JPEG 2000 Signature box, JP2 Header box, and one or more Contiguous Codestream boxes. A JP2 file may also contain other boxes as determined by the file writer. For example, a JP2 file may contain several XML boxes (containing metadata) between the JP2 Header box and the first Contiguous Codestream box.

I.2.3 Greyscale, colour, palette, multi-component specification

The JP2 file format provides two methods to specify the colourspace of the image. The enumerated method specifies the colourspace of an image by specifying a numeric value that specifies the colourspace. In this Recommendation | International Standard, images in the sRGB colourspace and greyscale images can be defined using the enumerated method.

The JP2 file format also provides for the specification of the colourspace of an image by embedding a restricted form of an ICC profile in the file. That profile shall be of either the Monochrome or Three-Component Matrix-Based class of input profiles as defined by the ICC Profile Format Specification, ICC.1:1998-09. This allows for the specification of a wide range of greyscale and RGB class colourspaces, as well as a few other spaces that can be represented by those two profile classes. See J.9 for a more detailed description of the legal colourspace transforms, how those transforms are stored in the file, and how to process an image using that transform without using an ICC colour management engine. While restricted, these ICC profiles are fully compliant ICC profiles and the image can thus be processed through any ICC compliant engine that supports profiles as defined in ICC.1:1998-09.

In addition to specifying the colourspace of the image, this Recommendation | International Standard provides a means by which a single component palettized image can be decoded and converted back to multiple-component form by the translation from index space to multiple-component space. Any such depalettization is applied before the colourspace is interpreted. In the case of palettized images, the specification of the colourspace of the image is applied to the multiple-component values stored in the palette.

I.2.4 Inclusion of opacity channels

The JP2 file format provides a means to indicate the presence of auxiliary channels (such as opacity), to define the type of those channels, and to specify the ordering and source of those channels (whether they are directly extracted from the codestream or generated by applying a palette to a codestream component). 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 colourspace specification. Once the file components have been mapped to the colour channels, the decompressed image can be processed through any needed colourspace transformations.

In many applications, components other than the colour channels 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 colour and auxiliary channels within a single codestream.

How applications deal with opacity or other auxiliary channels is outside the scope of this Recommendation | International Standard.

I.2.5 Metadata

One important aspect of the JP2 file format is the ability to add metadata to a JP2 file. Because all information is encapsulated in boxes, and all boxes have types, the format provides a simple mechanism for a reader to extract relevant 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 or private implementation. Also, any new box added to a JP2 file shall not change the visual appearance of the image.

I.2.6 Conformance with the file format

All conforming files shall contain all boxes required by this Recommendation | International Standard, and those boxes shall be as defined in this Recommendation | International Standard. Also, all conforming readers shall correctly interpret all required boxes defined in this Recommendation | International Standard and thus shall correctly interpret all conforming files.

I.3 Greyscale/Colour/Palettized/multi-component specification architecture

One of the most important aspects of a file format is that it specifies the colourspace of the contained image data. In order to properly display or interpret the image data, it is essential that the colourspace of that image is properly characterized. The JP2 file format provides a multi-level mechanism for characterizing the colourspace of an image.

I.3.1 Enumerated method

The simplest method for characterizing the colourspace of an image is to specify an integer code representing the colourspace in which the image is encoded. This method handles the specification of sRGB, greyscale, and sYCC images. Extensions to this method can be used to specify other colourspaces, including the definition of multi-component images.

For example, the image file may indicate that a particular image is encoded in the sRGB colourspace. To properly interpret and display the image, an application must natively understand the definition of the sRGB colourspace. Because an application must natively understand each specified colourspace, the complexity of this method is dependent on the exact colourspaces specified. Also, complexity of this mechanism is proportional to the number of colourspaces that are specified and required for conformance. While this method provides a high level of interoperability for images encoded using colourspaces for which correct interpretation is required for conformance, this method is very inflexible. This Recommendation | International Standard defines a specific set of colourspaces for which interpretation is required for conformance.

I.3.2 Restricted ICC profile method

An application may also specify the colourspace of an image using two restricted types of ICC profiles. This method handles the specification of the most commonly used RGB and greyscale class colourspaces through a low-complexity method.

An ICC profile is a standard representation of the transformation required to convert one colourspace into another colourspace. With respect to the JP2 file format, an ICC profile defines how decompressed samples from the codestream are converted into a standard colorspace (the Profile Connection Space (PCS)). Depending on the original colourspace of the samples, this transformation may be either very simple or very complex.

The ICC Profile Format Specification defines two specific classes of ICC profiles that are simple to implement, referred to within the profile specification as Monochrome Input and Three-Component Matrix-Based Input Profiles. These profiles limit the transformation from the source colourspace to the PCS_{XYZ} to the application of a non-linearity curve and a 3×3 matrix. It is practical to expect all applications, including simple devices, to be able to process the image through this transformation. Thus all conforming applications are required to correctly interpret the colourspace of any image that specifies the colourspace using this subset of possible ICC profile types.

For the JP2 file format, profiles shall conform to the ICC profile definition as defined by the ICC Profile Format Specification, ICC.1:1998-09, as well as the restrictions specified above. See J.9 for a more detailed description of the legal colourspace transforms, how those transforms are stored in the file, and how to process an image using that transform without using an ICC colour management engine.

I.3.3 Using multiple methods

Architecturally, the format allows for multiple methods to be embedded in a file and allows other standards to define additional enumerated methods and to define extended methods. This provides readers conforming to those extensions a choice as to what image processing path should be used to interpret the colourspace of the image. However, the first method found in the file (in the first Colourspace Specification box in the JP2 Header box) shall be one of the methods as defined and restricted in this Recommendation | International Standard. A conforming reader shall use that first method and ignore all other methods (in additional Colourspace Specification boxes) found in the file.

I.3.4 Palettized images

In addition to specifying the interpretation of the image in terms of colourspace, this Recommendation | International Standard allows for the decoding of a single component where the value of that single component represents an index into a palette of colours. Input of a decompressed sample to the palette converts the single value to a multiple-component tuple. The value of that tuple represents the colour of that sample; that tuple shall then be interpreted according to the other colour specification methods (Enumerated or Restricted ICC) as if that multiple-component sample had been directly extracted from multiple components in the codestream.

I.3.5 Interactions with the decorrelating multiple component transform

The specification of colour 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 A.5.1 and in Annex G). The colourspace transformations specified through the sequence of Colour Specification 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 colour image data.

I.3.6 Key to graphical descriptions (informative)

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

These descriptions are followed by a figure that shows the order and relationship of the parameters in the box. Figure I.2 shows an example of this type of figure. A rectangle is used to indicate the parameters in the box. The width of the rectangle is proportional to the number of bytes in the parameter. A shaded rectangle (diagonal stripes) indicates that the parameter is of varying size. Two parameters with superscripts and a grey area between indicate a run of several of these parameters. A sequence of two groups of multiple parameters with superscripts separated by a grey area indicates a run of that group of parameters (one set of each parameter in the group, followed by the next set of each parameter in the group). Optional parameters or boxes will be shown with a dashed rectangle.

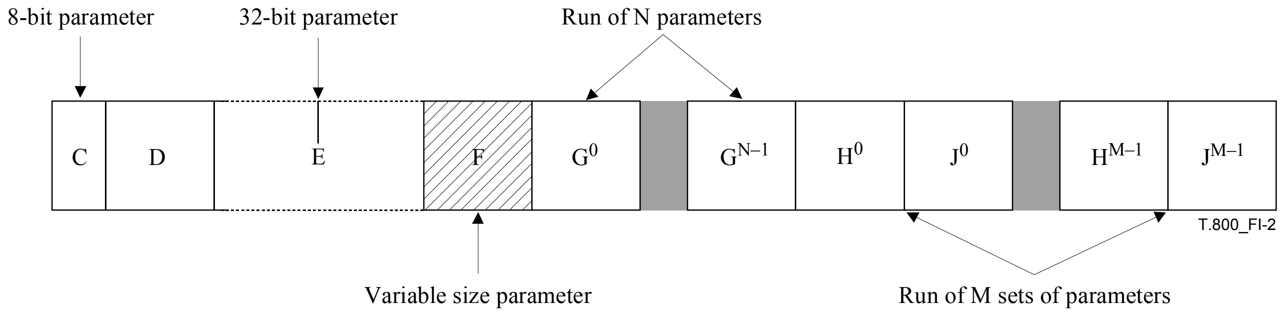


Figure I.2 – Example of the box description figures

The figure is followed by a list that describes the meaning of each parameter in the box. If parameters are repeate, the d length and nature of the run of parameters is defined. As an example, in Figure I.2, parameters C, D, E and F are 8-, 16-, 32-bit and variable length respectively. The notation G^0 and G^{N-1} implies that there are N different parameters, G^i, in a row. The group of parameters H^0 and H^{M-1}, and J^0 and J^{M-1} specify that the box will contain H^0, followed by J^0, followed by H^1 and J^1, continuing to H^{M-1} and J^{M-1} (M instances of each parameter in total). Also, the field E is optional and may not be found in this box.

After the list is a table that either describes the allowed parameter values or provides references to other tables that describe these values.

In addition, in a figure describing the contents of a superbox, an ellipsis (…) will be used to indicate that contents of the file between two boxes is not specifically defined. Any box (or sequence of boxes), unless otherwise specified by the definition of that box, may be found in place of the ellipsis.

For example, the superbox shown in Figure I.3 must contain an AA box and a BB box, and the BB box must follow the AA box. However, there may be other boxes found between boxes AA and BB. Dealing with unknown boxes is discussed in I.8.

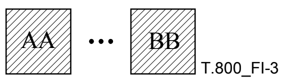


Figure I.3 – Example of the superbox description figures

I.4 Box definition

Physically, each object in the file is encapsulated within a binary structure called a box. That binary structure is as in Figure I.4:

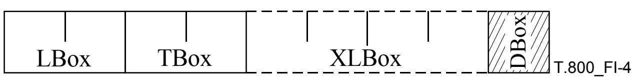


Figure I.4 – Organization of a box

LBox: Box Length. This field specifies the length of the box, stored as a 4-byte big endian 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 known when the LBox field was written. In this case, this box contains all bytes up to the end of the file. If a box of length 0 is contained within another box (its superbox), then the length of that superbox shall also be 0. This means that this box is the last box in the file. The values 2-7 are reserved for ISO use.

TBox: Box Type. This field specifies the type of information found in the DBox field. The value of this field is encoded as a 4-byte big endian unsigned integer. However, boxes are generally referred to by an ISO/IEC 646 character string translation of the integer value. For all box types defined within this Recommendation | International Standard, box types will be indicated as both character string (normative) and as 4-byte hexadecimal integers (informative). Also, a space character is shown in the character string translation of the box type as "\040". All values of TBox not defined within this Recommendation | International Standard are reserved for ISO use.

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 big endian unsigned integer. The value includes all of the fields of the box, including the LBox, TBox and XLBox fields.

DBox: Box Contents. This field contains the actual information contained within this box. The format of the box contents depends on the box type and will be defined individually for each type.

Table I.1 – Binary structure of a box

Field name	Size (bits)	Value
LBox	32	0, 1, or 8 to $(2^{32}-1)$
TBox	32	Variable
XLBox	64 0	16 to $(2^{64}-1)$; if LBox = 1 Not applicable; if LBox ≠ 1
DBox	Variable	Variable

For example, consider the illustration in Figure I.5 of a sequence of boxes, including one box that contains other boxes:

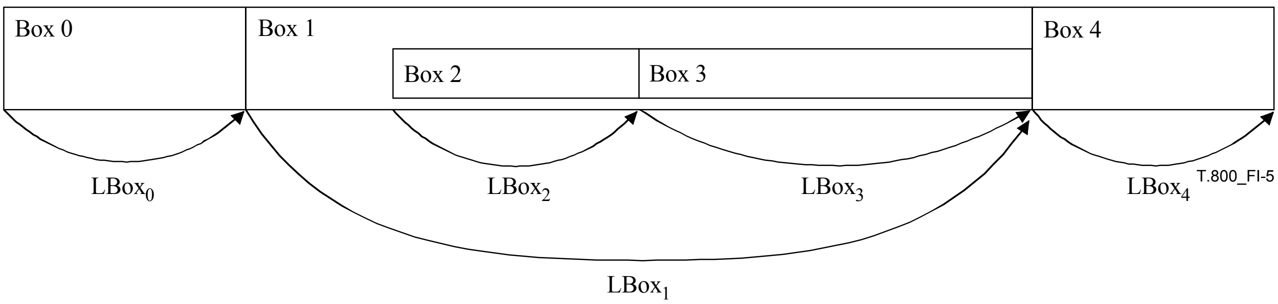


Figure I.5 – Illustration of box lengths

As shown in Figure I.5, 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 1 from the beginning of Box 1.

Table I.2 lists all boxes defined by this Recommendation | International Standard. Indentation within the table indicates the hierarchical containment structure of the boxes within a JP2 file.

Table I.2 – Defined boxes

Box name	Type	Superbox	Required?	Comments
JPEG 2000 Signature box	'jP\040\040' (0x6A50 2020)	No	Required	This box uniquely identifies the file as being part of the JPEG 2000 family of files.
File Type box	'ftyp' (0x6674 7970)	No	Required	This box specifies file type, version and compatibility information, including specifying if this file is a conforming JP2 file or if it can be read by a conforming JP2 reader.
JP2 Header box	'jp2h' (0x6A70 3268)	Yes	Required	This box contains a series of boxes that contain header-type information about the file.
Image Header box	'ihdr' (0x6968 6472)	No	Required	This box specifies the size of the image and other related fields.
Bits Per Component box	'bpcc' (0x6270 6363)	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.
Colour Specification box	'colr' (0x636F 6C72)	No	Required	This box specifies the colourspace of the image.
Palette box	'pclr' (0x7063 6C72)	No	Optional	This box specifies the palette which maps a single component in index space to a multiple-component image.
Component Mapping box	'cmap' (0x636D 6170)	No	Optional	This box specifies the mapping between a palette and codestream components.
Channel Definition box	'cdef' (0x6364 6566)	No	Optional	This box specifies the type and ordering of the components within the codestream, as well as those created by the application of a palette.
Resolution box	'res\040' (0x7265 7320)	Yes	Optional	This box contains the grid resolution.
Capture Resolution box	'resc' (0x7265 7363)	No	Optional	This box specifies the grid resolution at which the image was captured.
Default Display Resolution box	'resd' (0x7265 7364)	No	Optional	This box specifies the default grid resolution at which the image should be dis played.
Contiguous Codestream box	'jp2c' (0x6A70 3263)	No	Required	This box contains the codestream as defined by Annex A.
Intellectual Property box	'jp2i' (0x6A70 3269)	No	Optional	This box contains intellectual property information about the image.
XML box	'xml\040' (0x786D 6C20)	No	Optional	This box provides a tool by which vendors can add XML formatted information to a JP2 file.
UUID box	'uuid' (0x7575 6964)	No	Optional	This box provides a tool by which vendors can add additional information to a file without risking conflict with other vendors.
UUID Info box	'uinf' (0x7569 6E66)	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' (0x7563 7374)	No	Optional	This box specifies a list of UUIDs.
URL box	'url\040' (0x7572 6C20)	No	Optional	This box specifies a URL.

I.5 Defined boxes

The following boxes shall properly be interpreted by all conforming readers. Each of these boxes conforms to the standard box structure as defined in I.4. The following clauses define the value of the DBox field from Table I.1 (the contents of the box). It is assumed that the LBox, TBox and XLBox fields exist for each box in the file as defined in Annex I.4.

I.5.1 JPEG 2000 Signature box

The JPEG 2000 Signature box identifies that the format of this file was defined by the JPEG 2000 Recommendation | International Standard, as well as provides a small amount of information which can help determine the validity of the rest of the file. The JPEG 2000 Signature box shall be the first box in the file, and all files shall contain one and only one JPEG 2000 Signature box.

The type of the JPEG 2000 Signature box shall be 'jP\040\040' (0x6A50 2020). The length of this box shall be 12 bytes. The contents of this box shall be the 4-byte character string '<CR><LF><0x87><LF>' (0x0D0A 870A). For file verification purposes, this box can be considered a fixed-length 12-byte string which shall have the value: 0x0000 000C 6A50 2020 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 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.

I.5.2 File Type box

The File Type box specifies the Recommendation | International Standard which completely defines all of the contents of this file, as well as a separate list of readers, defined by other Recommendations | International Standards, with which this file is compatible, and thus the file can be properly interpreted within the scope of that other standard. This box shall immediately follow the JPEG 2000 Signature box. This differentiates between the standard which completely describes the file, from other standards that interpret a subset of the file.

All files shall contain one and only one File Type box.

The type of the File Type Box shall be 'ftyp' (0x6674 7970). The contents of this box shall be as in Figure I.6:

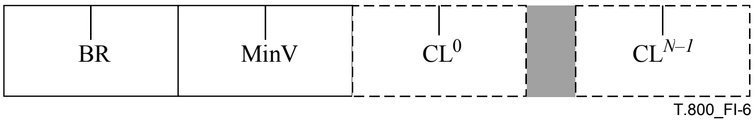


Figure I.6 – Organization of the contents of a File Type box

BR: Brand. This field specifies the Recommendation | International Standard which completely defines this file. This field is specified by a four-byte string of ISO/IEC 646 characters. The value of this field is defined in Table I.3:

Table I.3 – Legal Brand values

Value	Meaning
'jp2\040'	IS 15444-1, Annex I (This Recommendation | International Standard)
other values	Reserved for other ISO uses

In addition, the Brand field shall be considered functionally equivalent to a major version number. A major version change (if there ever is one), representing an incompatible change in the JP2 file format, shall define a different value for the Brand field.

If the value of the Brand field is not 'jp2\040', then a value of 'jp2\040' in the Compatibility list indicates that a JP2 reader can interpret the file in some manner as intended by the creator of the file.

MinV: Minor version. This parameter defines the minor version number of this JP2 specification for which the file complies. The parameter is defined as a 4-byte big endian unsigned integer. The value of this field shall be zero. However, readers shall continue to parse and interpret this file even if the value of this field is not zero.

CL^i: Compatibility list. This field specifies a code representing this Recommendation | International Standard, another standard, or a profile of another standard, to which the file conforms. This field is encoded as a four-byte string of ISO/IEC 646 characters. A file that conforms to this Recommendation | International Standard shall have at least one CL^i field in the File Type box, and shall contain the value 'jp2\040' in one of the CL^i fields in the File Type box, and all conforming readers shall properly interpret all files with 'jp2\040' in one of the CL^i fields

If one of the CL^i fields contains the value "J2P0" then the first codestream contained within this JP2 file is restricted as described for Profile-0 from Table A.45.

If one of the CL^i fields contains the value "J2P1" then the first codestream contained within this JP2 file is restricted as described for Profile-1 from Table A.45.

Other values of the Compatibility list field are reserved for ISO use.

The number of CL^i fields is determined by the length of this box.

Table I.4 – Format of the contents of the File Type box

Field name	Size (bits)	Value
BR	32	0 to $(2^{32}-1)$
MinV	32	0
CL^i	32	0 to $(2^{32}-1)$

I.5.3 JP2 Header box (superbox)

The JP2 Header box contains generic information about the file, such as number of components, colourspace, and grid resolution. This box is a superbox. Within a JP2 file, there shall be one and only one JP2 Header box. The JP2 Header box may be located anywhere within the file after the File Type box but before the Contiguous Codestream box. It also must be at the same level as the JPEG 2000 Signature and File Type boxes (it shall not be inside any other superbox within the file).

The type of the JP2 Header box shall be 'jp2h' (0x6A70 3268).

This box contains several boxes. Other boxes may be defined in other standards and may be ignored by conforming readers. Those boxes contained within the JP2 Header box that are defined within this Recommendation | International Standard are as in Figure I.7:

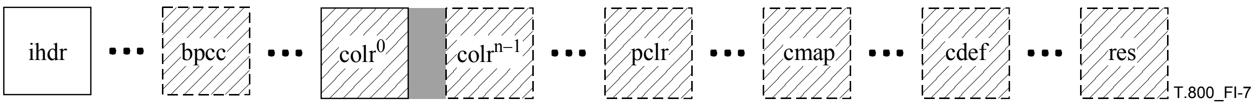


Figure I.7 – Organization of the contents of a JP2 Header box

ihdr: Image Header box. This box specifies information about the image, such as its height and width. Its structure is specified in I.5.3.1. This box shall be the first box in the JP2 Header box.

bpcc: Bits Per Component box. This box specifies the bit depth of each component in the codestream after decompression. Its structure is specified in I.5.3.2. This box may be found anywhere in the JP2 Header box provided that it comes after the Image Header box.

$colr^i$: Colour Specification boxes. These boxes specify the colourspace of the decompressed image. Their structures are specified in I.5.3.3. There shall be at least one Colour Specification box within the JP2 Header box. The use of multiple Colour Specification boxes provides the ability for a decoder to be given multiple optimization or compatibility options for colour processing. These boxes may be found anywhere in the JP2 Header box provided that they come after the Image Header box. All Colour Specification boxes shall be contiguous within the JP2 Header box.

pclr: Palette box. This box defines the palette to use to create multiple components from a single component. Its structure is specified in I.5.3.4. This box may be found anywhere in the JP2 Header box provided that it comes after the Image Header box.

cmap: Component Mapping box. This box defines how image channels are identified from the actual components in the codestream. Its structure is specified in I.5.3.5. This box may be found anywhere in the JP2 Header box provided that it comes after the Image Header box.

cdef: Channel Definition box. This box defines the channels in the image. Its structure is specified in I.5.3.6. This box may be found anywhere in the JP2 Header box provided that it comes after the Image Header box.

res: Resolution box. This box specifies the capture and default display grid resolutions of the image. Its structure is specified in I.5.3.7. This box may be found anywhere in the JP2 Header box provided that it comes after the Image Header box.

I.5.3.1 Image Header box

This box contains fixed length generic information about the image, such as the image size and number of components. The contents of the JP2 Header box shall start with an Image Header box. Instances of this box in other places in the file shall be ignored. The length of the Image Header box shall be 22 bytes, including the box length and type fields. Much of the information within the Image Header box is redundant with information stored in the codestream itself.

All references to "the codestream" in the descriptions of fields in this Image Header box apply to the codestream found in the first Contiguous Codestream box in the file. Files that contain contradictory information between the Image Header box and the first codestream are not conforming files. However, readers may choose to attempt to read these files by using the values found within the codestream.

The type of the Image Header box shall be 'ihdr' (0x6968 6472) and contents of the box shall have the format as in Figure I.8:

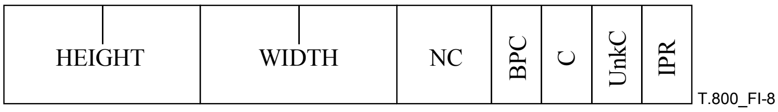


Figure I.8 – Organization of the contents of an Image Header box

HEIGHT: Image area height. The value of this parameter indicates the height of the image area. This field is stored as a 4-byte big endian unsigned integer. The value of this field shall be Ysiz – YOsiz, where Ysiz and YOsiz are the values of the respective fields in the SIZ marker in the codestream. See Figure B.1 for an illustration of the image area. However, reference grid points are not necessarily square; the aspect ratio of a reference grid point is specified by the Resolution box. If the Resolution box is not present, then a reader shall assume that reference grid points are square.

WIDTH: Image area width. The value of this parameter indicates the width of the image area. This field is stored as a 4-byte big endian unsigned integer. The value of this field shall be Xsiz – XOsiz, where Xsiz and XOsiz are the values of the respective fields in the SIZ marker in the codestream. See Figure B.1 for an illustration of the image area. However, reference grid points are not necessarily square; the aspect ratio of a reference grid point is specified by the Resolution box. If the Resolution box is not present, then a reader shall assume that reference grid points are square.

NC: Number of components. This parameter specifies the number of components in the codestream and is stored as a 2-byte big endian unsigned integer. The value of this field shall be equal to the value of the Csiz field in the SIZ marker in the codestream.

BPC: Bits per component. This parameter specifies the bit depth of the components in the codestream, minus 1, and is stored as a 1-byte field.

If the bit depth and the sign are the same for all components, then this parameter specifies that bit depth and shall be equivalent to the values of the $Ssiz^i$ fields in the SIZ marker in the codestream (which shall all be equal). If the components vary in bit depth and/or sign, then the value of this field shall be 255 and the JP2 Header box shall also contain a Bits Per Component box defining the bit depth of each component (as defined in I.5.3.2).

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. Other values are reserved for ISO use.

UnkC: Colourspace Unknown. This field specifies if the actual colourspace of the image data in the codestream is known. This field is encoded as a 1-byte unsigned integer. Legal values for this field are 0, if the colourspace of the image is known and correctly specified in the Colourspace Specification boxes within the file, or 1, if the colourspace of the image is not known. A value of 1 will be used in cases such as the transcoding of legacy images where the actual colourspace of the image data is not known. In those cases, while the colourspace 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 ISO use.

IPR: Intellectual Property. This parameter indicates whether this JP2 file contains intellectual property rights information. If the value of this field is 0, this file does not contain rights information, and thus the file does not contain an IPR box. If the value is 1, then the file does contain rights information and thus does contain an IPR box as defined in I.6. Other values are reserved for ISO use.

Table I.5 – Format of the contents of the Image Header box

Field name	Size (bits)	Value
HEIGHT	32	1 to $(2^{32}-1)$
WIDTH	32	1 to $(2^{32}-1)$
NC	16	1 to 16 384
BPC	8	See Table I.6
C	8	7
Unk	8	0 to 1
IPR	8	0 to 1

Table I.6 – BPC values

Values (bits) MSB LSB	Component sample precision

`x000 0000`
to
`x010 0101`
	Component bit depth = value + 1. From 1 bit deep through 38 bits deep respectively (counting the sign bit, if appropriate)

`0xxx xxxx`	Components are unsigned values
`1xxx xxxx`	Components are signed values
`1111 1111`	Components vary in bit depth
	All other values reserved for ISO use

I.5.3.2 Bits Per Component box

The Bits Per Component box specifies the bit depth of each component. If the bit depth of all components in the codestream is the same (in both sign and precision), then this box shall not be found. Otherwise, this box specifies the bit depth of each individual component. The order of bit depth values in this box is the actual order in which 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.

There shall be one and only one Bits Per Component box inside a JP2 Header box.

The type of the Bits Per Component Box shall be 'bpcc' (0x6270 6363). The contents of this box shall be as in Figure I.9:

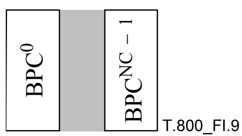


Figure I.9 – Organization of the contents of a Bits Per Component box

BPC^i: Bits per component. This parameter specifies the bit depth of component *i*, minus 1, encoded as a 1-byte value. The ordering of the components within the Bits Per Component 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 value of this field shall be equivalent to the respective $Ssiz^i$ field in the SIZ marker in the codestream.

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 I.7 – Format of the contents of the Bits Per Component box

Field name	Size (bits)	Value
BPC^i	8	See Table I.8

Table I.8 – BPC^i values

Values (bits) MSB LSB	Component sample precision
x000 0000 to x010 0101	Component bit depth = value + 1. From 1 bit deep through 38 bits deep respectively (counting the sign bit, if appropriate)
0xxx xxxx	Components are unsigned values
1xxx xxxx	Components are signed values
	All other values reserved for ISO use

I.5.3.3 Colour Specification box

Each Colour Specification box defines one method by which an application can interpret the colourspace of the decompressed image data. This colour 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 image data.

A JP2 file may contain multiple Colour Specification boxes, but must contain at least one, specifying different methods for achieving "equivalent" results. A conforming JP2 reader shall ignore all Colour Specification boxes after the first. However, readers conforming to other standards may use those boxes as defined in those other standards.

The type of a Colour Specification box shall be 'colr' (0x636F 6C72). The contents of a Colour Specification box is as in Figure I.10:

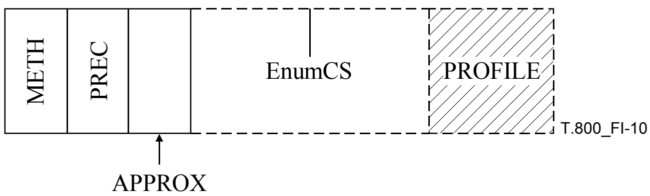


Figure I.10 – Organization of the contents of a Colour Specification box

METH: Specification method. This field specifies the method used by this Colour Specification box to define the colourspace of the decompressed image. This field is encoded as a 1-byte unsigned integer. The value of this field shall be 1 or 2, as defined in Table I.9.

Table I.9 – Legal METH values

Value	Meaning
1	**Enumerated Colourspace**. This colourspace specification box contains the enumerated value of the colourspace 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	**Restricted ICC profile**.This Colour Specification box contains an ICC profile in the PROFILE field. This profile shall specify the transformation needed to convert the decompressed image data into the PCS_{XYZ}, and shall conform to either the Monochrome Input or Three-Component Matrix-Based Input profile class, and contain all the required tags specified therein, as defined in ICC.1:1998-09. As such, the value of the Profile Connection Space field in the profile header in the embedded profile shall be 'XYZ\040' (0x5859 5A20) indicating that the output colourspace of the profile is in the XYZ colourspace. Any private tags in the ICC profile shall not change the visual appearance of an image processed using this ICC profile. The components from the codestream may have a range greater than the input range of the tone reproduction curve (TRC) of the ICC profile. Any decoded values should be clipped to the limits of the TRC before processing the image through the ICC profile. For example, negative sample values of signed components may be clipped to zero before processing the image data through the profile. See J.9 for a more detailed description of the legal colourspace transforms, how those transforms are stored in the file, and how to process an image using that transform without using an ICC colour management engine. 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.
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, and a conforming JP2 reader shall ignore the entire Colour Specification box.

PREC: Precedence. This field is reserved for ISO use and the value shall be set to zero; however, conforming readers shall ignore the value of this field. This field is specified as a signed 1-byte integer.

APPROX: Colourspace approximation. This field specifies the extent to which this colour specification method approximates the "correct" definition of the colourspace. The value of this field shall be set to zero; however, conforming readers shall ignore the value of this field. Other values are reserved for other ISO use. This field is specified as 1-byte unsigned integer.

EnumCS: Enumerated colourspace. This field specifies the colourspace of the image using integer codes. To correctly interpret the colour of an image using an enumerated colourspace, the application must know the definition of that colourspace internally. This field contains a 4-byte big endian unsigned integer value indicating the colourspace of the image. If the value of the METH field is 2, then the EnumCS field shall not exist. Valid EnumCS values for the first colourspace specification box in conforming files are limited to 16, 17, and 18 as defined in Table I.10:

Table I.10 – Legal EnumCS values

Value	Meaning
16	sRGB as defined by IEC 61966-2-1
17	greyscale: A greyscale space where image luminance is related to code values using the sRGB non-linearity given in Equations (2) through (4) of IEC 61966-2-1 (sRGB) specification: $$Y' = Y_{8\,bit} / 255 \quad \text{(I-1)}$$ $$\text{for } (Y' \le 0{,}04045),\ Y_{lin} = Y' / 12{,}92 \quad \text{(I-2)}$$ $$\text{for } (Y' > 0{,}04045),\ Y_{lin} = \left(\frac{Y' + 0{,}055}{1{,}055}\right)^{2{,}4}$$ where Y_{lin} is the linear image luminance value in the range 0.0 to 1.0. The image luminance values should be interpreted relative to the reference conditions in Section 2 of IEC 61966-2-1.
18	sYCC as defined by IEC 61966-2-1 Amd. 1 NOTE – It is not recommend to use ICT or RCT specified in Annex G with sYCC image data. See J.15 for guidelines on handling YCC codestreams.
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 ICC.1:1998-09.

Table I.11 – Format of the contents of the Colour Specification box

Field name	Size (bits)	Value
METH	8	1 to 2
PREC	8	0
APPROX	8	0
EnumCS	32 if METH = 1 0 if METH = 2	0 to (2^{32}– 1) no value
PROFILE	Variable	Variable; see the ICC Profile Format Specification, version ICC.1:1998-09.

I.5.3.4 Palette box

This box specifies a palette that can be used to create channels from components. However, the Palette box does not specify the creation of any particular channel; the creation of channels based on the application of the palette to a component is specified by the Component Mapping box. The colourspace or meaning of the generated channel is specified by the Channel Definition box (or specified through the defaults defined in the specification of the Channel Definition box if the Channel Definition box does not exist). If the JP2 Header box contains a Palette box, then it shall also contain a Component Mapping box. If the JP2 Header box does not contain a Palette box, then it shall not contain a Component Mapping box.

There shall be at most one Palette box inside a JP2 Header box.

The type of the Palette box shall be 'pclr' (0x7063 6C72). The contents of this box shall be as in Figure I.11:

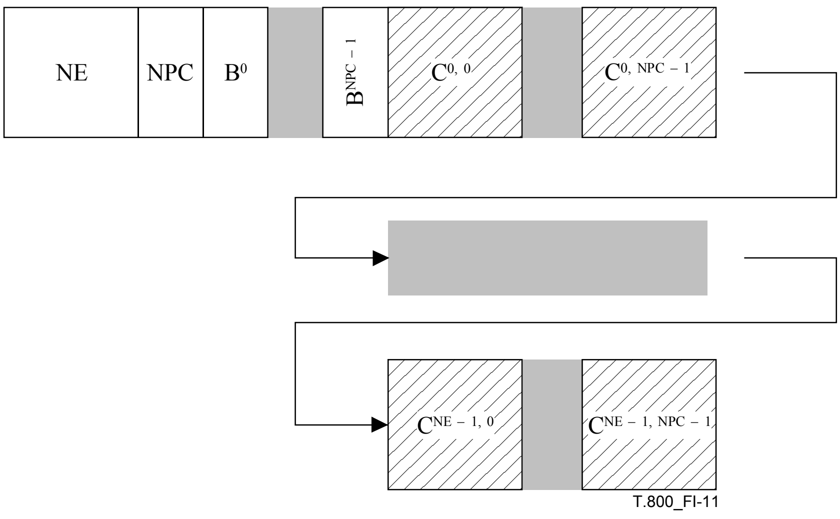


Figure I.11 – Organization of the contents of the Palette box

NE: Number of entries in the table. This value shall be in the range 1 to 1024 and is encoded as a 2-byte big endian unsigned integer.

NPC: Number of palette columns specified in the Palette box. For example, if the palette is to be used to map a single index component into a three-component RGB image, then the value of this field shall be 3. This field is encoded as a 1-byte unsigned integer.

B^i: This parameter specifies the bit depth of values created by palette column *i*, encoded as a 1-byte big endian integer. The low 7-bits of the value indicate the bit depth of this palette column. The high-bit indicates whether the palette column is signed or unsigned. If the high-bit is 1, then the palette column contains signed values. If the high-bit is 0, then the palette column contains unsigned values. The number of B^i values shall be the same as the value of the NPC field.

C^{ji}: The value for entry *j* for palette column *i*. C^{ji} values are organized in component major order; all of the values for entry *j* are grouped together, followed by all of the values for entry $j + 1$. In the example given above, this table would therefore read R_1, G_1 ,B_1 ,R_2 ,G_2 ,B_2, etc. The size of C^{ji} is the value specified by field B^i. The number of palette columns shall be the same as the NPC field. The number of C^{ji} values shall be the number of palette columns (the NPC field) times the number of entries in the palette (NE). If the value of B^i is not a multiple of 8, then each C^{ji} value is padded with zeros to a multiple of 8 bits and the actual value shall be stored in the low-order bits of the padded value. For example, if the value of B^i is 10 bits, then the individual C^{ji} values shall be stored in the low 10 bits of a 16-bit field.

Table I.12 – Format of the contents of the Palette box

Field name	Size (bits)	Value
NE	16	1 to 1024
NPC	8	1 to 255
B^i	8	See Table I.13
C^{ji}	Variable	Variable

Table I.13 – B^i values

Values (bits) MSB LSB	Palette column sample precision

`x000 0000`
to
`x010 0101`
	Palette column bit depth = value + 1. From 1 bit deep through 38 bits deep respectively (counting the sign bit, if appropriate)

`0xxx xxxx`	Palette column values are unsigned values
`1xxx xxxx`	Palette column values are signed values
	All other values reserved for ISO use.

I.5.3.5 Component Mapping box

The Component Mapping box defines how image channels are identified from the actual components decoded from the codestream. This abstraction allows a single structure (the Channel Definition box) to specify the colour or type of both palettized images and non-palettized images. This box contains an array of CMP^i, $MTYP^i$ and $PCOL^i$ fields. Each group of these fields represents the definition of one channel in the image. The channels are numbered in order starting with zero, and the number of channels specified in the Component Mapping box is determined by the length of the box.

There shall be at most one Component Mapping box inside a JP2 Header box.

If the JP2 Header box contains a Palette box, then the JP2 Header box shall also contain a Component Mapping box. If the JP2 Header box does not contain a Component Mapping box, the components shall be mapped directly to channels, such that component *i* is mapped to channel *i*.

The type of the Component Mapping box shall be 'cmap' (0x636D 6170). The contents of this box shall be as in Figure I.12:

Figure I.12 – Organization of the contents of a Component Mapping box

CMP^i: This field specifies the index of component from the codestream that is mapped to this channel (either directly or through a palette). This field is encoded as a 2-byte big endian unsigned integer.

$MTYP^i$: This field specifies how this channel is generated from the actual components in the file. This field is encoded as a 1-byte unsigned integer. Legal values of the $MTYP^i$ field are as in Table I.14:

Table I.14 – $MTYP^i$ field values

Value	Meaning
0	Direct use. This channel is created directly from an actual component in the codestream. The index of the component mapped to this channel is specified in the CMP^i field for this channel.
1	Palette mapping. This channel is created by applying the palette to an actual component in the codestream. The index of the component mapped into the palette is specified in the CMP^i field for this channel. The column from the palette to use is specified in the $PCOL^i$ field for this channel.
2 to 255	Reserved for ISO use

$PCOL^i$: This field specifies the index component from the palette that is used to map the actual component from the codestream. This field is encoded as a 1-byte unsigned integer. If the value of the $MTYP^i$ field for this channel is 0, then the value of this field shall be 0.

Table I.15 – Format of the contents of the Component Mapping box

Field name	Size (bits)	Value
CMP^i	16	0 to 16 384
$MTYP^i$	8	0 to 1
$PCOL^i$	8	0 to 255

I.5.3.6 Channel Definition box

The Channel Definition box specifies the meaning of the samples in each channel in the image. The exact location of this box within the JP2 Header box may vary provided that it follows the Image Header box.The mapping between actual components from the codestream to channels is specified in the Component Mapping box. If the JP2 Header box does not contain a Component Mapping box, then a reader shall map component *i* to channel *i*, for all components in the codestream.

There shall be at most one Channel Definition box inside a JP2 Header box.

This box contains an array of channel descriptions. For each description, three values are specified: the index of the channel described by that association, the type of that channel, and the association of that channel with particular colours. This box may specify multiple descriptions for a single channel.

If a multiple component transform is specified within the codestream, the image must be in an RGB colourspace and the red, green and blue colours as channels 0, 1 and 2 in the codestream, respectively.

The type of the Channel Definition box shall be 'cdef' (0x6364 6566). The contents of this box shall be as in Figure I.13:

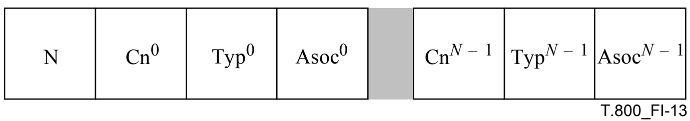


Figure I.13 – Organization of the contents of a Channel Definition box

N: Number of channel descriptions. This field specifies the number of channel descriptions in this box. This field is encoded as a 2-byte big endian unsigned integer.

Cn^i: Channel index. This field specifies the index of the channel for this description. The value of this field represents the index of the channel as defined within the Component Mapping box (or the actual component from the codestream if the file does not contain a Component Mapping box). This field is encoded as a 2-byte big endian unsigned integer.

Typ^i: Channel type. This field specifies the type of the channel for this description. The value of this field specifies the meaning of the decompressed samples in this channel. This field is encoded as a 2-byte big endian unsigned integer. Legal values of this field are shown in Table I.16:

Table I.16 – Typ^i field values

Value	Meaning
0	This channel is the colour image data for the associated colour.
1	Opacity. A sample value of 0 indicates that the sample is 100% transparent, and the maximum value of the channel (related to the bit depth of the codestream component or the related palette component mapped to this channel) indicates a 100% opaque sample. All opacity channels shall be mapped from unsigned components.
2	Premultiplied opacity. An opacity channel as specified above, except that the value of the opacity channel has been multiplied into the colour channels for which this channel is associated. Premultiplication is defined as follows: $$S_p = S \times \frac{\alpha}{\alpha_{max}} \quad \text{(I-3)}$$ where S is the original sample, S_p is the pre multiplied sample (the sample stored in the image, α is the value of the opacity channel, and α_{max} is the maximum value of the opacity channel as defined by the bit depth of the opacity channel.
3 to (2^{16} – 2)	Reserved for ISO use
2^{16} – 1	The type of this channel is not specified.

$Asoc^i$: Channel association. This field specifies the index of the colour for which this channel is directly associated (or a special value to indicate the whole image or the lack of an association). For example, if this channel is an opacity channel for the red channel in an RGB colourspace, this field would specify the index of the colour red. Table I.17 specifies legal association values. Table I.18 specifies legal colour indices. This field is encoded as a 2-byte big endian unsigned integer.

Table I.17 – $Asoc^i$ field values

Value	Meaning
0	This channel is associated as the image as a whole (for example, an independent opacity channel that should be applied to all color channels).
1 to (2^{16} – 2)	This channel is associated with a particular colour as indicated by this value. This value is used to associate a particular channel with a particular aspect of the specification of the colourspace of this image. For example, indicating that a channel is associated with the red channel of an RGB image allows the reader to associate that decoded channel with the Red input to an ICC profile contained within a Colour Specification box. Colour indicators are specified in Table I.18.
2^{16} – 1	This channel is not associated with any particular colour.

Table I.18 – Colours indicated by the Asoc^i field

Class of colourspace	Colour indicated by the following value of the Asoc^i field			
	1	**2**	**3**	**4**
RGB	R	G	B	
Greyscale	Y			
YC_bC_r	Y	C_b	C_r	
The following colourspace classes are listed for future reference, as well as to aid in under standing 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 colour colourspaces	1	2	3	4

The values in Table I.18 specify indices that have been assigned to represent specific "colours" and do not refer to specific channels (or components within the codestream or palette). Readers must use the information contained within the Channel Definition box to determine which channels contain which colours.

In this box, channel indices are mapped from particular components within the codestream or palette. Colour indices specify how a particular channel shall be interpreted based on the specification of the colourspace of the image. There shall be one channel definition in this box for every colour required by the colourspace specification of this file as specified by the Colourspace Specification box.

For example, the green colour in an RGB image is specified by a {Cn, Typ, Asoc} value of {*i*, 0, 2}, where *i* is the index of that channel (either directly or as generated by applying the reverse multiple component transform to the actual components in the codestream). Applications that are only concerned with extracting the colour channels can treat the Typ/Asoc field pair as a four-byte value where the combined value maps directly to the colour indices (as the Typ field for a colour channel shall be 0).

In another example, the codestream may contain a channel *i* that specifies opacity blending samples for the red and green channels, and a channel *j* that specifies opacity blending samples for the blue channel. In that file, the following {Cn, Typ, Asoc} tuples would be found in the Channel Definition box for the two opacity channels: {*i*, 1, 1}, {*i*, 1, 2} and {*j*, 1, 3}.

There shall not be more than one channel in a JP2 file with a the same Typ^i and Asoc^i value pair, with the exception of Typ^i and Asoc^i values of $2^{16}– 1$ (not specified). For example a JP2 file in an RGB colourspace shall only contain one green channel, and a greyscale image shall contain only one grey channel. There shall be either exactly one opacity channel, exactly one pre-multiplied opacity channel, or neither associated with a single colour channel in an image.

If the codestream contains only colour channels and those channels are ordered in the same order as the associated colours (for example, an RGB image with three channels in the order R, G, then B), then this box shall not exist. If there are any auxiliary channels or the channels are not in the same order as the colour indices, then the Channel Definition box (see Table I.19) shall be found within the JP2 Header box with a complete list of channel definitions.

Table I.19 – Format of the Channel Definition box

Parameter	Size (bits)	Value
N	16	1 to (2^{16}– 1)
Cn^i	16	0 to (2^{16}– 1)
Typ^i	16	0 to (2^{16}– 1)
$Asoc^i$	16	0 to (2^{16}– 1)

I.5.3.7 Resolution box (superbox)

This box specifies the capture and default display grid resolutions of this image. If this box exists, it shall contain either a Capture Resolution box, or a Default Display Resolution box, or both.

There shall be at most one Resolution box inside a JP2 Header box.

The type of a Resolution box shall be 'res\040' (0x7265 7320). The contents of the Resolution box are as in Figure I.14:

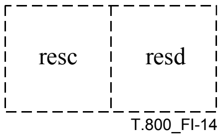


Figure I.14 – Organization of the contents of the Resolution box

resc: Capture Resolution box. This box specifies the grid resolution at which this image was captured. The format of this box is specified in I.5.3.7.1.

resd: Default Display Resolution box. This box specifies the default grid resolution at which this image should be displayed. The format of this box is specified in I.5.3.7.2

I.5.3.7.1 Capture Resolution box

This box specifies the grid 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 grid resolution could also specify the resolution of an aerial digital camera or satellite camera.

The vertical and horizontal capture grid resolutions are calculated using the six parameters (Table I.20) stored in this box in the following two equations, respectively:

$$VRc = \frac{VRcN}{VRcD} \times 10^{VRcE} \tag{I-4}$$

$$HRc = \frac{HRcN}{HRcD} \times 10^{HRcE} \tag{I-5}$$

The values *VRc* and *HRc* are always in reference grid points per meter. If an application requires the grid resolution in another unit, then that application must apply the appropriate conversion.

The type of a Capture Resolution box shall be 'resc' (0x7265 7363). The contents of the Capture Resolution box are as in Figure I.15:

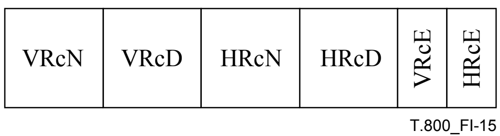


Figure I.15 – Organization of the contents of the Capture Resolution box

VRcN: Vertical Capture grid resolution numerator. This parameter specifies the *VRcN* value in Equation (I-4), which is used to calculate the vertical capture grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

VRcD: Vertical Capture grid resolution denominator. This parameter specifies the *VRcD* value in Equation (I-4), which is used to calculate the vertical capture grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

HRcN: Horizontal Capture grid resolution numerator. This parameter specifies the *HRcN* value in Equation (I-5), which is used to calculate the horizontal capture grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

HRcD: Horizontal Capture grid resolution denominator. This parameter specifies the *HRcD* value in Equation (I-5), which is used to calculate the horizontal capture grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

VRcE: Vertical Capture grid resolution exponent. This parameter specifies the *VRcE* value in Equation (I-4), which is used to calculate the vertical capture grid resolution. This parameter is encoded as a twos-complement 1-byte signed integer.

HRcE: Horizontal Capture grid resolution exponent. This parameter specifies the *HRcE* value in Equation (I-5), which is used to calculate the horizontal capture grid resolution. This parameter is encoded as a twos-complement 1-byte signed integer.

Table I.20 – Format of the contents of the Capture Resolution box

Field name	Size (bits)	Value
VRcN	16	1 to $(2^{16}-1)$
VRcD	16	1 to $(2^{16}-1)$
HRcN	16	1 to $(2^{16}-1)$
HRcD	16	1 to $(2^{16}-1)$
VRcE	8	–128 to 127
HRcE	8	–128 to 127

I.5.3.7.2 Default Display Resolution box

This box specifies a desired display grid resolution. 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. However, this value is only a default. Each application must determine an appropriate display size for that application.

The vertical and horizontal display grid resolutions are calculated using the six parameters (Table I.21) stored in this box in the following two equations, respectively:

$$VRd = \frac{VRdN}{VRdD} \times 10^{VRdE} \quad \text{(I-6)}$$

$$HRd = \frac{HRdN}{HRdD} \times 10^{HRdE} \quad \text{(I-7)}$$

The values *VRd* and *HRd* are always in reference grid points per meter. If an application requires the grid resolution in another unit, then that application must apply the appropriate conversion.

The type of a Default Display Resolution box shall be 'resd' (0x7265 7364). The contents of the Default Display Resolution box are as in Figure I.16:

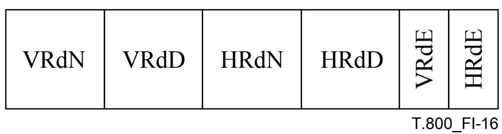


Figure I.16 – Organization of the contents of the Default Display Resolution box

VRdN: Vertical Display grid resolution numerator. This parameter specifies the *VRdN* value in Equation (I-6), which is used to calculate the vertical display grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

VRdD: Vertical Display grid resolution denominator. This parameter specifies the *VRdD* value in Equation (I-6), which is used to calculate the vertical display grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

HRdN: Horizontal Display grid resolution numerator. This parameter specifies the *HRdN* value in Equation (I-7), which is used to calculate the horizontal display grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

HRdD: Horizontal Display grid resolution denominator. This parameter specifies the *HRdD* value in Equation (I-7), which is used to calculate the horizontal display grid resolution. This parameter is encoded as a 2-byte big endian unsigned integer.

VRdE: Vertical Display grid resolution exponent. This parameter specifies the *VRdE* value in Equation (I-6), which is used to calculate the vertical display grid resolution. This parameter is encoded as a twos-complement 1-byte signed integer.

HRdE: Horizontal Display grid resolution exponent. This parameter specifies the *HRdE* value in Equation (I-7), which is used to calculate the horizontal display grid resolution. This parameter is encoded as a twos-complement 1-byte signed integer.

Table I.21 – Format of the contents of the Default Display Resolution box

Field name	Size (bits)	Value
VRdN	16	1 to (2^{16}– 1)
VRdD	16	1 to (2^{16}– 1)
HRdN	16	1 to (2^{16}– 1)
HRdD	16	1 to (2^{16}– 1)
VRdE	8	–128 to 127
HRdE	8	–128 to 127

I.5.4 Contiguous Codestream box

The Contiguous Codestream box contains a valid and complete JPEG 2000 codestream, as defined in Annex A. When displaying the image, a conforming reader shall ignore all codestreams after the first codestream found in the file. Contiguous Codestream boxes may be found anywhere in the file except before the JP2 Header box.

The type of a Contiguous Codestream box shall be 'jp2c' (0x6A70 3263). The contents of the box shall be as in Figure I.17:

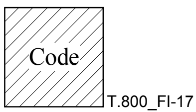


Figure I.17 – Organization of the contents of the Contiguous Codestream box

Code: This field contains a valid and complete JPEG 2000 codestream as specified by Annex A.

Table I.22 – Format of the contents of the Contiguous Codestream box

Field name	Size (bits)	Value
Code	Variable	Variable

I.6 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 in this Recommendation | International Standard as a means to allow applications to recognize the existence of IPR information. Use and interpretation of this information is beyond the scope of this Recommendation | International Standard.

In general, an IPR box found at the top level of the file specifies IPR for the file as a whole. IPR boxes may be found at other locations, including inside superboxes defined by other Recommendations | International Standards. For those IPR boxes, the rights specified refer to the entity defined by the containing superbox.

The type of the Intellectual Property Box shall be 'jp2i' (0x6A70 3269).

I.7 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 to the JP2 file format. All of the following boxes are optional in conforming files and may be ignored by conforming readers.

I.7.1 XML boxes

An XML box contains vendor-specific information (in XML format) other than the information contained within boxes defined by 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 File Type box.

The type of an XML box is 'xml\040' (0x786D 6C20). The contents of the box shall be as in Figure I.18:

T.800_FI-18

Figure I.18 – Organization of the contents of the XML box

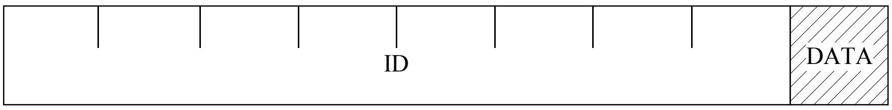
DATA: This field shall contain a well-formed XML document as defined by REC-xml-19980210.

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 Recommendation | International Standard, and the correct interpretation of the contents of any XML box shall not change the visual appearance of the image. All readers may ignore any XML box in the file.

I.7.2 UUID boxes

A UUID box contains vendor-specific information other than the information contained within boxes 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 File Type box.

The type of a UUID box shall be 'uuid' (0x7575 6964). The contents of the box shall be as in Figure I.19:



T.800_FI-19

Figure I.19 – Organization of the contents of the UUID box

ID: This field contains a 16-byte UUID as specified by ISO/IEC 11578. The value of this UUID specifies the format of the vendor-specific information stored in the DATA field and the interpretation of that information.

DATA: This field contains the vendor-specific information. The format of this information is defined outside of the scope of this Recommendation | International Standard, but is indicated by the value of the UUID field.

Table I.23 – Format of the contents of a UUID box

Field name	Size (bits)	Value
UUID	128	Variable
DATA	Variable	Variable

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 information in any UUID box shall not change the visual appearance of the image. All readers may ignore any UUID box.

I.7.3 UUID Info boxes (superbox)

While it is useful to allow vendors to extend JP2 files by adding information 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 File Type box.

These boxes, if present, may not provide a complete index for the UUIDs in the file, may reference UUIDs 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' (0x7569 6E66). The contents of a UUID Info box are as in Figure I.20:

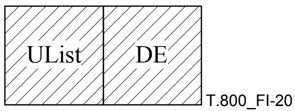


Figure I.20 – Organization of the contents of a UUID Info box

UList: UUID List box. This box contains a list of UUIDs for which this UUID Info box specifies a link to more information. The format of the UUID List box is specified in I.7.3.1.

DE: Data Entry URL box. This box contains a URL. An application can acquire more information about the UUIDs contained in the UUID List box. The format of a Data Entry URL box is specified in I.7.3.2

I.7.3.1 UUID List box

This box contains a list of UUIDs. The type of a UUID List box shall be 'ulst' (0x756C 7374). The contents of a UUID List box shall be as in Figure I.21:

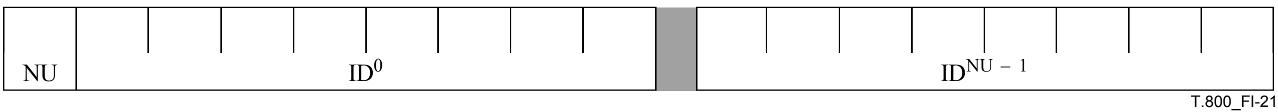


Figure I.21 – Organization of the contents of a UUID List box

NU: Number of UUIDs. This field specifies the number of UUIDs found in this UUID List box. This field is encoded as a 2-byte big endian unsigned integer.

ID^i: ID. This field specifies one UUID, as specified in ISO/IEC 11578, which shall be associated with the URL contained in the URL 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 I.24 – UUID List box contents data structure values

Parameter	Size (bits)	Value
NU	16	0 to (2^{16}– 1)
$UUID^i$	128	0 to (2^{128}– 1)

I.7.3.2 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 information 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., URLs of type file, http, ftp, etc.), which ideally also permits random access. Relative URLs are permissible and are relative to the file containing this Data Entry URL box.

The type of a Data Entry URL box shall be 'url\040' (0x7572 6C20). The contents of a Data Entry URL box shall be as in Figure I.22:

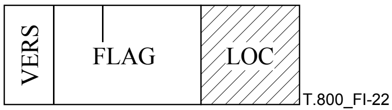


Figure I.22 – Organization of the contents of a Data Entry URL box

VERS: Version number. This field specifies the version number of the format of this box and is encoded as a 1-byte unsigned integer. The value of this field shall be 0.

FLAG: Flags. This field is reserved for other use to flag particular attributes of this box and is encoded as a 3-byte unsigned integer. The value of this field shall be 0.

LOC: Location. This field specifies the URL of the additional information associated with the UUIDs 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 I.25 – Data Entry URL box contents data structure values

Parameter	Size (bits)	Value
VERS	8	0
FLAG	24	0
LOC	varies	varies

I.8 Dealing with unknown boxes

A conforming JP2 file may contain boxes not known to applications based solely on this Recommendation | International Standard. If a conforming reader finds a box that it does not understand, it shall skip and ignore that box.

Annex J

Examples and guidelines

(This annex does not form an integral part of this Recommendation | International Standard)

This annex includes a number of examples intended to indicate how the encoding process works, and how the resulting codestream should be output. This annex is entirely informative.

J.1 Software conventions adaptive entropy decoder

This annex provides some alternate flowcharts for a version of the adaptive entropy decoder. This alternate version may be more efficient when implemented in software, as it has fewer operations along the fast path.

The alternate version is obtained by making the following substitutions. Replace the flowchart in Figure C.20 with the flowchart in Figure J.1. Replace the flowchart in Figure C.15 with the flowchart in Figure J.2. Replace the flowchart in Figure C.19 with the flowchart in Figure J.3.

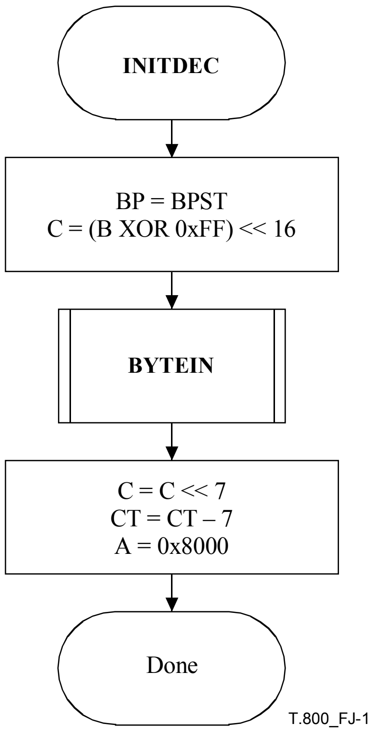


Figure J.1 – Initialization of the software-conventions decoder

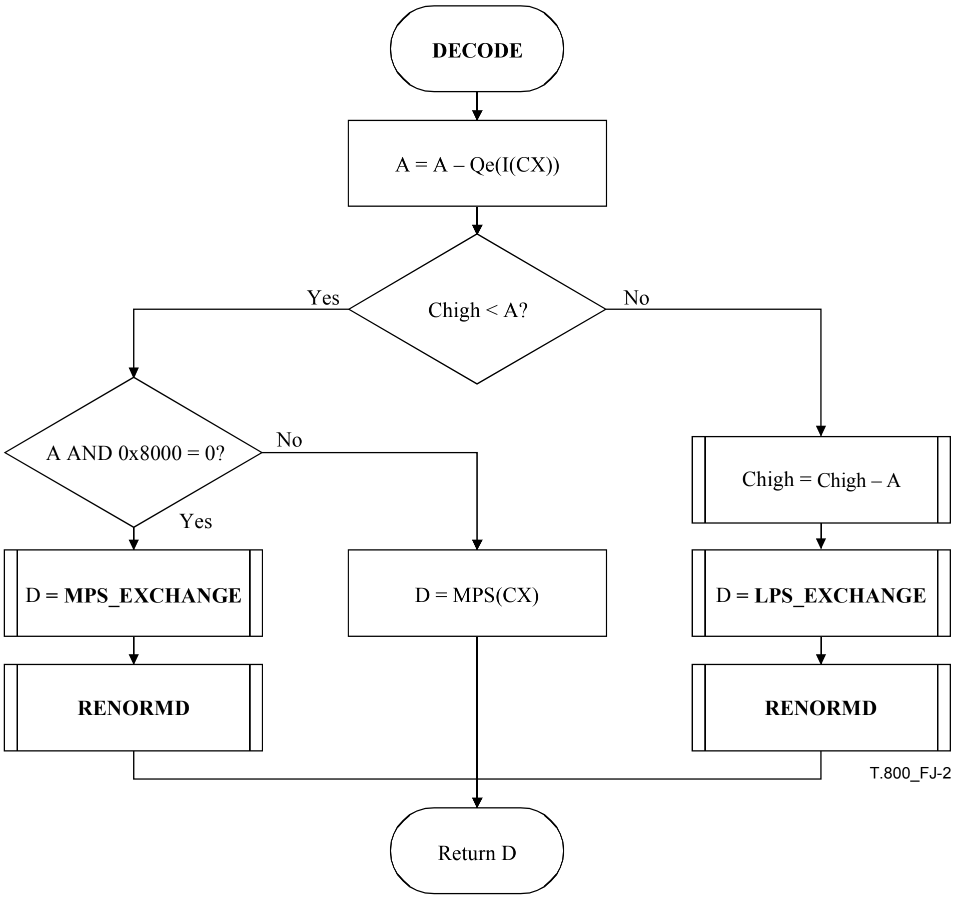


Figure J.2 – Decoding an MPS or an LPS in the software-conventions decoder

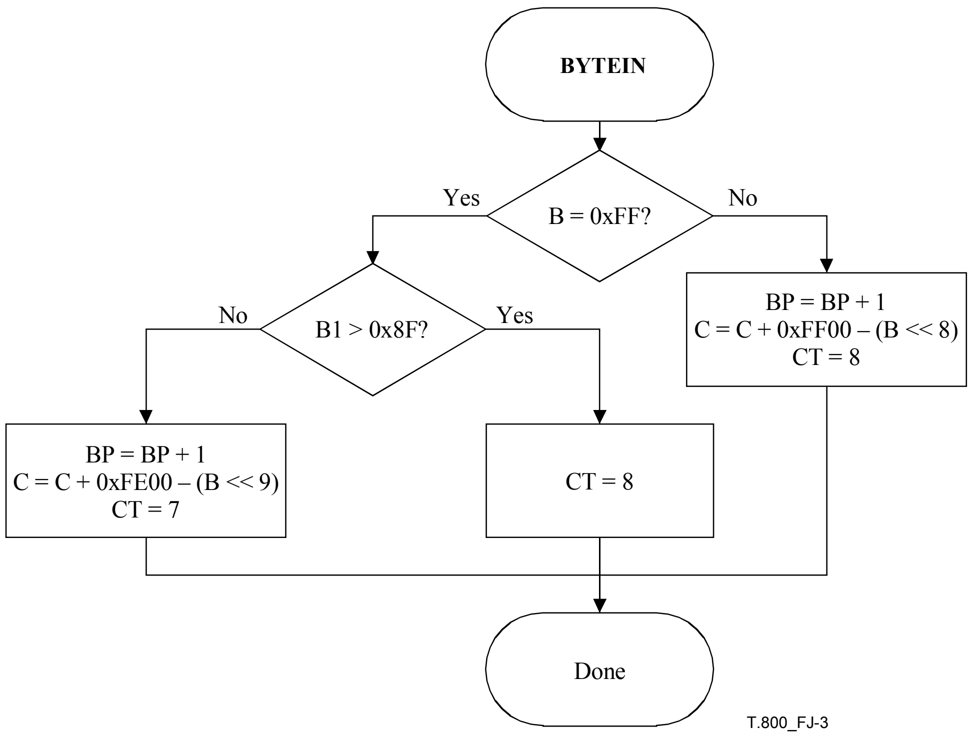


Figure J.3 – Inserting a new byte into the C register in the software-conventions decoder

J.2 Selection of quantization step sizes for irreversible transformations

For irreversible compression, no particular selection of the quantization step size is required in this Recommendation | International Standard. Different applications may specify the quantization step sizes according to specific tile-component characteristics. One effective way of selecting the quantizer step size for each sub-band *b* is to scale a default step size Δ_d by taking into account the horizontal and vertical filtering procedures which produced these sub-band coefficients. One method consists in scaling Δ_d with an energy weight parameter γ_b (the amount of squared errors introduced by a unit error in a transformed coefficient of sub-band *b*) in the following way [12]:

$$\Delta_b = \frac{\Delta_d}{\sqrt{\gamma_b}} \tag{J-1}$$

J.3 Filter impulse responses corresponding to lifting-based irreversible filtering procedures

The irreversible filtering procedures described in Annex F implement the 9-tap/7-tap Cohen-Daubechies-Feauveau convolutional filter bank [20], [21]. Equivalent impulse responses of the analysis and synthesis filters are given in Tables J.1 and J.2.

Table J.1 – Definition of impulse responses for the 9-7 irreversible analysis filter bank

n	**Low-pass filter**	**Approximate value**
0	$-5x_1\left(48\lvert x_2\rvert^2 - 16\,\Re x_2 + 3\right)/32$	0.602 949 018 236 360
±1	$-5x_1\left(8\lvert x_2\rvert^2 - \Re x_2\right)/8$	0. 266 864 118 442 875
±2	$-5x_1\left(4\lvert x_2\rvert^2 - 4\Re x_2 - 1\right)/16$	–0.078 223 266 528 990
±3	$-5x_1(\Re x_2)/8$	–0.016 864 118 442 875
±4	$-5x_1/64$	0.026 748 757 410 810
n	**High-pass filter**	**Approximate value**
–1	$(6x_1 - 1)/8x_1$	1.115 087 052 457 000
–2, 0	$-(16x_1 - 1)/32x_1$	–0.591 271 763 114 250
–3, 1	$(2x_1 + 1)/16x_1$	–0.057 543 526 228 500
–4, 2	$-1/32x_1$	0.091 271 763 114 250

Table J.2 – Definition of impulse responses for the 9-7 irreversible synthesis filter band

n	Low-pass filter	Approximate value
0	$(6x_1 - 1) / 8x_1$	1.115 087 052 457 000
±1	$(16x_1 - 1) / 32x_1$	0.591 271 763 114 250
±2	$(2x_1 + 1) / 16x_1$	–0.057 543 526 228 500
±3	$1 / 32x_1$	–0.091 271 763 114 250
n	**High-pass filter**	**Approximate value**
1	$-5x_1\left(48\lvert x_2\rvert^2 - 16\,\Re x_2 + 3\right) / 32$	0.602 949 018 236 360
0, 2	$5x_1\left(8\lvert x_2\rvert^2 - \Re x_2\right) / 8$	–0.266 864 118 442 875
–1, 3	$-5x_1\left(4\lvert x_2\rvert^2 - 4\Re x_2 - 1\right) / 16$	–0.078 223 266 528 990
–2, 4	$5x_1(\Re x_2) / 8$	0.016 864 118 442 875
–3, 5	$-5x_1 / 64$	0.026 748 757 410 811

J.4 Example of discrete wavelet transformation

Table J.3 contains the integer-valued samples $I(x, y)$ of a tile component that is 13 samples wide and 17 samples high.

Table J.3 – Source tile component samples

***I*(*x*, *y*)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
0	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	2	2	3	4	5	6	7	8	9	10	11	12
3	3	3	3	4	5	5	6	7	8	9	10	11	12
4	4	4	4	5	5	6	7	8	8	9	10	11	12
5	5	5	5	5	6	7	7	8	9	10	11	12	13
6	6	6	6	6	7	7	8	9	10	10	11	12	13
7	7	7	7	7	8	8	9	9	10	11	12	13	13
8	8	8	8	8	8	9	10	10	11	12	12	13	14
9	9	9	9	9	9	10	10	11	12	12	13	14	15
10	10	10	10	10	10	11	11	12	12	13	14	14	15
11	11	11	11	11	11	12	12	13	13	14	14	15	16
12	12	12	12	12	12	13	13	13	14	15	15	16	16
13	13	13	13	13	13	13	14	14	15	15	16	17	17
14	14	14	14	14	14	14	15	15	16	16	17	17	18
15	15	15	15	15	15	15	16	16	17	17	18	18	19
16	16	16	16	16	16	16	17	17	17	18	18	19	20

J.4.1 Example of 9-7 irreversible wavelet transformation

Tables J.4, J.5, J.6, J.7, J.8, J.9 and J.10 contain the coefficients of the sub-bands 2LL, 2HL, 2LH, 2HH, 1HL, 1LH, 1HH resulting from the two-level decomposition with the 9-7 irreversible transformation of the source tile component samples given in Table J.3 (see Figure F.18). The coefficients' values displayed in the tables have been rounded to the nearest integer.

Table J.4 – 2LL sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{2LL}(u, v)$	0	1	2	3
0	1	4	8	11
1	4	5	8	11
2	8	9	11	13
3	12	12	14	16
4	15	15	17	18

Table J.5 – 2HL sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{2HL}(u, v)$	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

Table J.6 – 2LH sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{2LH}(u, v)$	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Table J.7 – 2HH sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{2HH}(u, v)$	0	1	2
0	–1	0	0
1	0	1	0
2	0	0	0
3	0	0	0

Table J.8 – 1HL sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{1HL}(u, v)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	–1	0	0	0	0
4	0	0	0	–1	0	0
5	0	0	1	1	0	–1
6	0	0	0	0	0	0
7	0	0	–1	–1	–1	0
8	0	0	0	0	0	0

Table J.9 – 1LH sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{1LH}(u, v)$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	–1	0	0	0
2	0	0	0	0	0	1	1
3	0	0	0	0	–1	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Table J.10 – 1HH sub-band coefficients (9-7 irreversible wavelet transformation)

$a_{1HH}(u, v)$	0	1	2	3	4	5
0	–1	0	0	0	0	0
1	0	0	–1	0	0	0
2	0	–1	1	0	0	0
3	0	0	0	0	0	1
4	0	0	0	0	–1	0
5	0	0	0	0	0	0
6	0	0	–1	0	–1	1
7	0	0	0	0	–1	0

J.4.2 Example of 5-3 reversible wavelet transformation

Tables J.11, J.12, J.13, J.14, J.15, J.16 and J.17 contain the coefficients of the subbands 2LL, 2HL, 2LH, 2HH, 1HL, 1LH, 1HH resulting from the two-level decomposition with the 5-3 irreversible transformation of the source tile component samples given in Table J.3 (see Figure F.18). The coefficients' values displayed in the tables have been rounded to the nearest integer.

Table J.11 – 2LL sub-band coefficients (5-3 reversible wavelet transformation)

$a_{2HL}(u, v)$	0	1	2	3
0	0	4	8	12
1	4	5	8	12
2	8	8	11	15
3	12	12	14	18
4	16	16	18	20

Table J.12 – 2HL sub-band coefficients (5-3 reversible wavelet transformation)

$a_{2HL}(u, v)$	0	1	2
0	0	0	0
1	0	1	0
2	0	1	0
3	0	0	1
4	0	0	0

Table J.13 – 2LH sub-band coefficient (5-3 reversible wavelet transformation)

$a_{2LH}(u, v)$	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	0	0	0
3	0	0	0	0

Table J.14 – 2HH sub-band coefficients (5-3 reversible wavelet transformation)

$a_{2HH}(u, v)$	0	1	2
0	–1	0	0
1	0	–1	0
2	0	1	0
3	0	0	0

Table J.15 – 1HL sub-band coefficients (5-3 reversible wavelet transformation)

$a_{1HL}(u, v)$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	1	0	1	0	0
3	0	0	0	0	–1	1
4	0	0	0	0	1	1
5	0	0	1	1	0	–1
6	0	0	1	0	1	1
7	0	0	0	0	0	0
8	0	0	0	0	0	0

Table J.16 – 1LH sub-band coefficients (5-3 reversible wavelet transformation)

$a_{1LH} (u, v)$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	0	0	0	1	1
3	0	0	1	0	0	1	1
4	0	0	0	0	1	0	2
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1
7	0	0	0	0	1	1	0

Table J.17 – 1HH sub-band coefficients (5-3 reversible wavelet transformation)

a_{1HH} (*u*, *v*)	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	1	0	1	0
3	0	0	0	0	0	1
4	0	0	0	0	0	1
5	0	0	0	1	0	0
6	0	0	0	0	0	1
7	0	0	0	0	–1	0

J.5 Row-based wavelet transform

Described here is an example of a row-based wavelet transformation for the 9-7 irreversible filter well suited for compression devices which receive and transfer image data in a serial manner. Traditional wavelet transformation implementations require the whole image to be buffered, and then the 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 transformation implementation. The row-based wavelet transformation overcomes the previous limitation while providing the exact same transformed coefficients as a traditional wavelet transformation implementation. However, the row-based wavelet transformation alone does not provide a complete row-based encoding paradigm. A complete row-based coder also has to take into account all the following coding stages up to the entropy coding and rate allocation stages.

J.5.1 The FDWT_ROW procedure

The FDWT_ROW procedure for the 9-7 irreversible filter uses one buffer *buf*(*i*, *j*) of five rows, $0 \le i \le 4$, to perform the equivalent of the 2D_SD procedure described in F.4.2, except for the 2D_DEINTERLEAVE procedure. The range of the samples of input tile component *I*(*x*, *y*) is assumed to be defined by Equation (F-1). Each row of the buffer *buf*(*i*, *j*) is of size $tcx_1 - tcx_0 + 1$. The general description of the FDWT_ROW applied to one image tile component is illustrated in Figure J.4 for the first level of decomposition. The FDWT_ROW takes as input a level shifted image tile component row of samples and produces as output one row of transform coefficients. It is assumed throughout this clause that the image tile component has at least five rows.

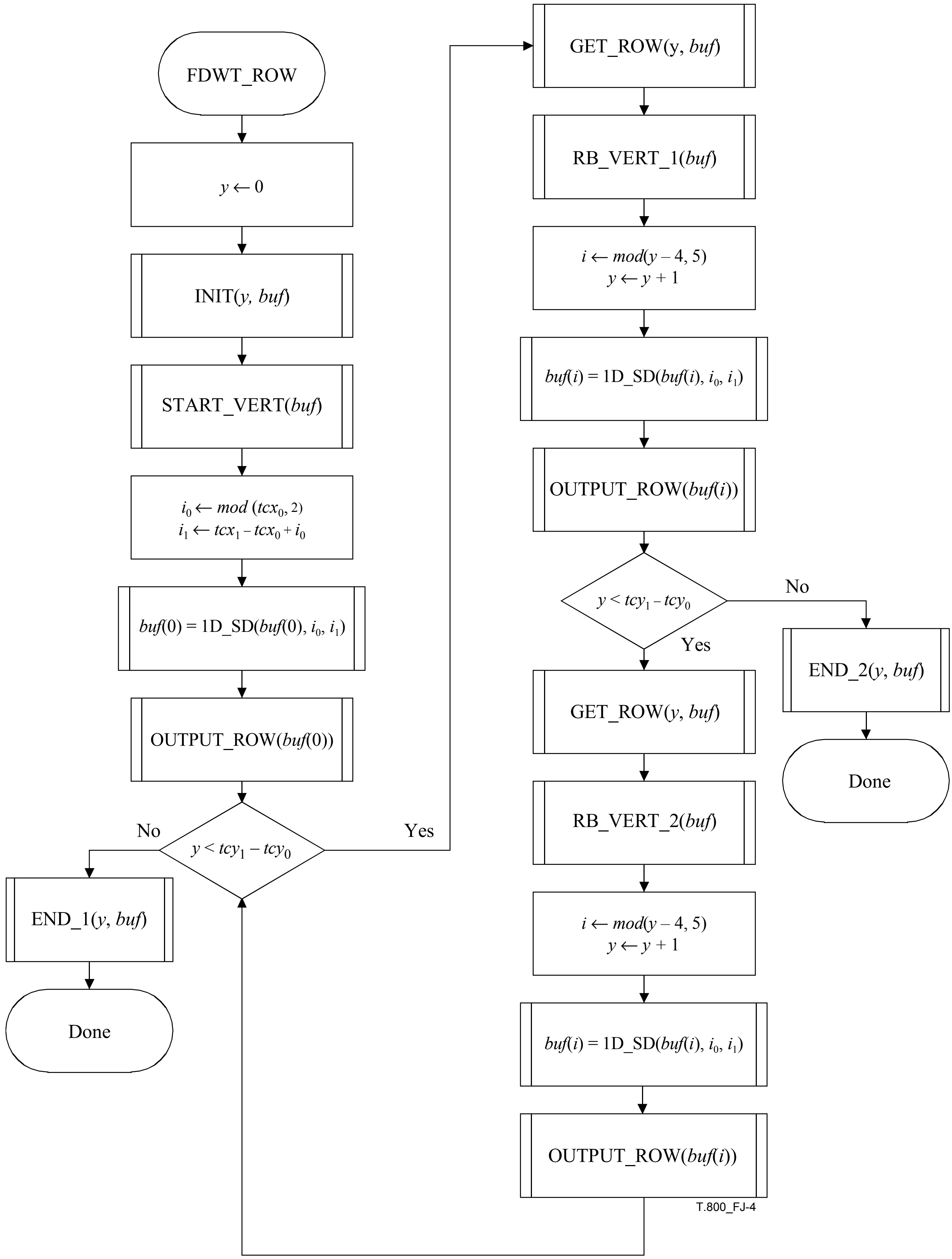


Figure J.4 – The FDWT_ROW procedure

J.5.1.1 The GET_ROW procedure

In this description, the level shifted image tile component is assumed to be stored in an external memory $I(x, y)$. As illustrated in Figure J.5, the GET_ROW procedure reads one row of samples of the level shifted image tile component and transfers this row of samples in the buffer, *buf*.

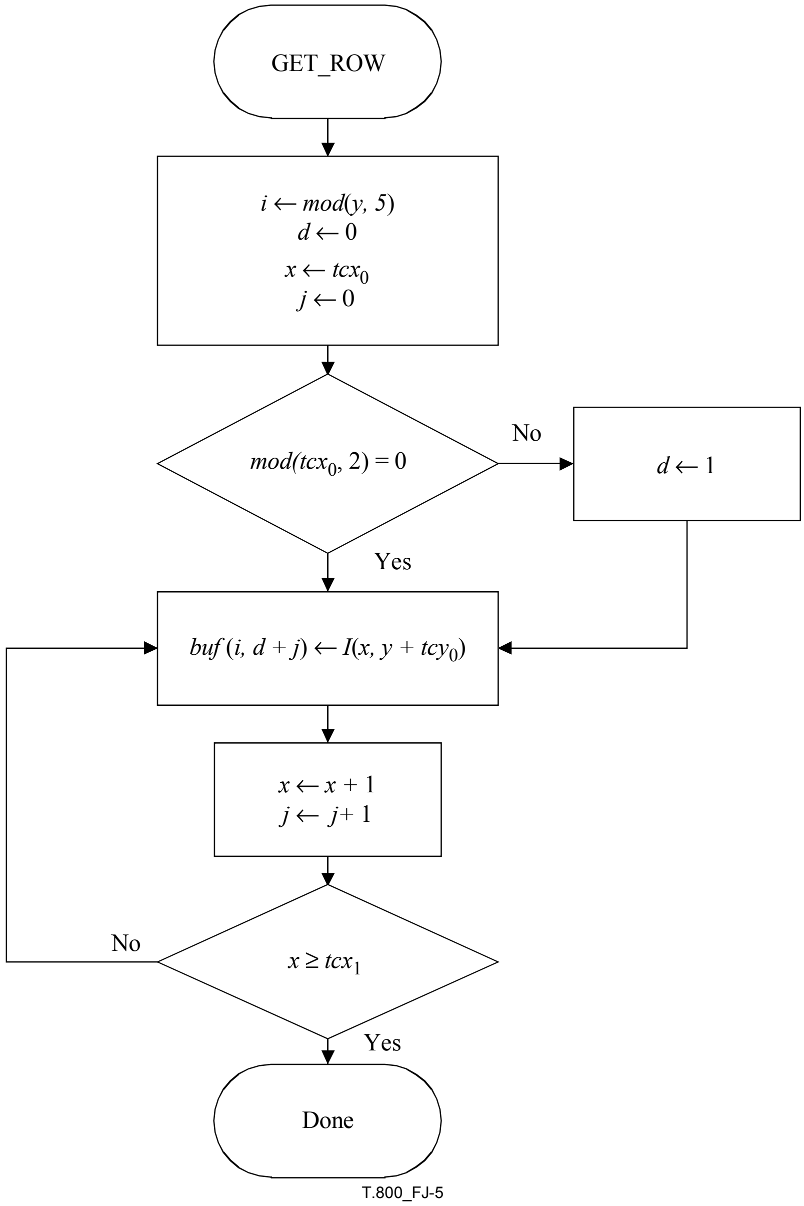


Figure J.5 – The GET_ROW procedure

J.5.2 The INIT procedure

As illustrated in Figure J.6, the INIT procedure reads five rows of samples of the level shifted image tile component and transfers these rows of samples to the buffer, *buf*.

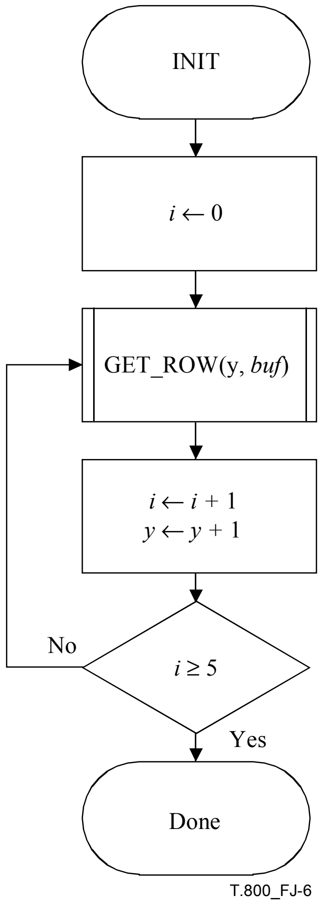


Figure J.6 – The INIT procedure

J.5.3 The START_VERT procedure

As illustrated in Figure J.7, the START_VERT procedure modifies the coefficients in the buffer *buf*(i, j). In this figure as well as in all the following figures of this clause, 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 $d \le j < tcx_0 - tcx_1 + d$.

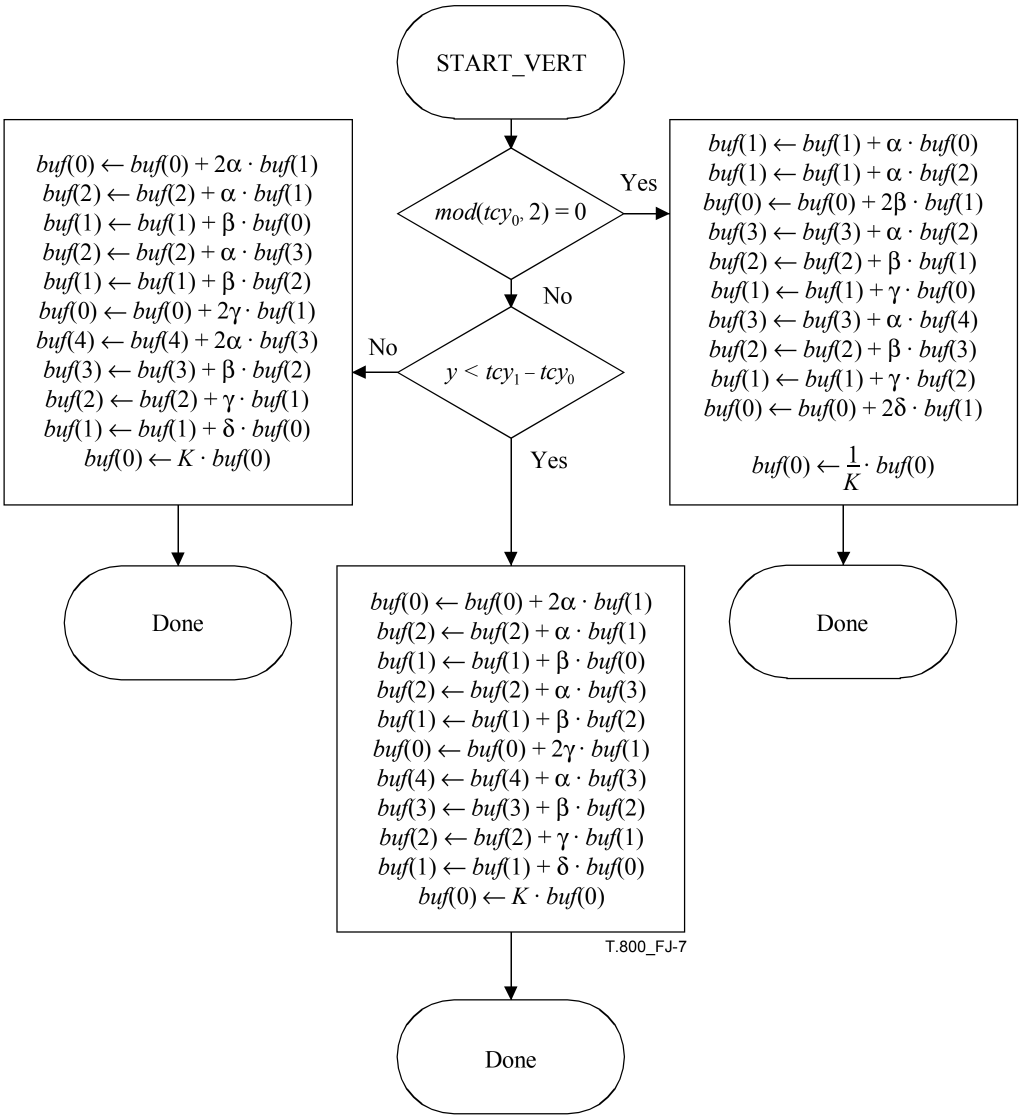


Figure J.7 – The START_VERT procedure

J.5.3.1 The RB_VERT_1 procedure

As illustrated in Figure J.8, the RB_VERT_1 procedure modifies the coefficients in *buf*(i, j).

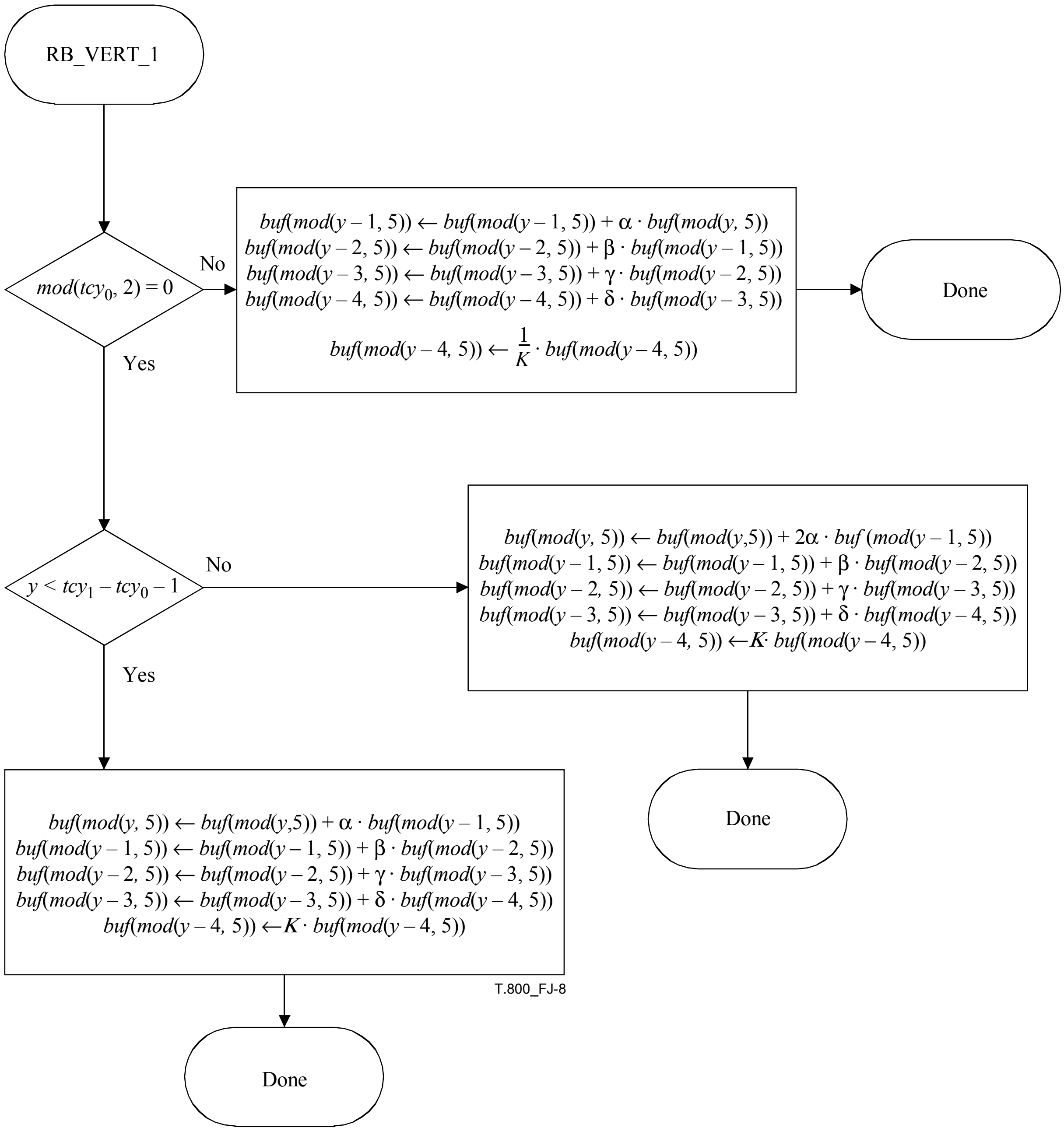


Figure J.8 – The RB_VERT_1 procedure

J.5.3.2 The RB_VERT_2 procedure

As illustrated in Figure J.9, the RB_VERT_2 procedure modifies the coefficients in $buf(i, j)$.

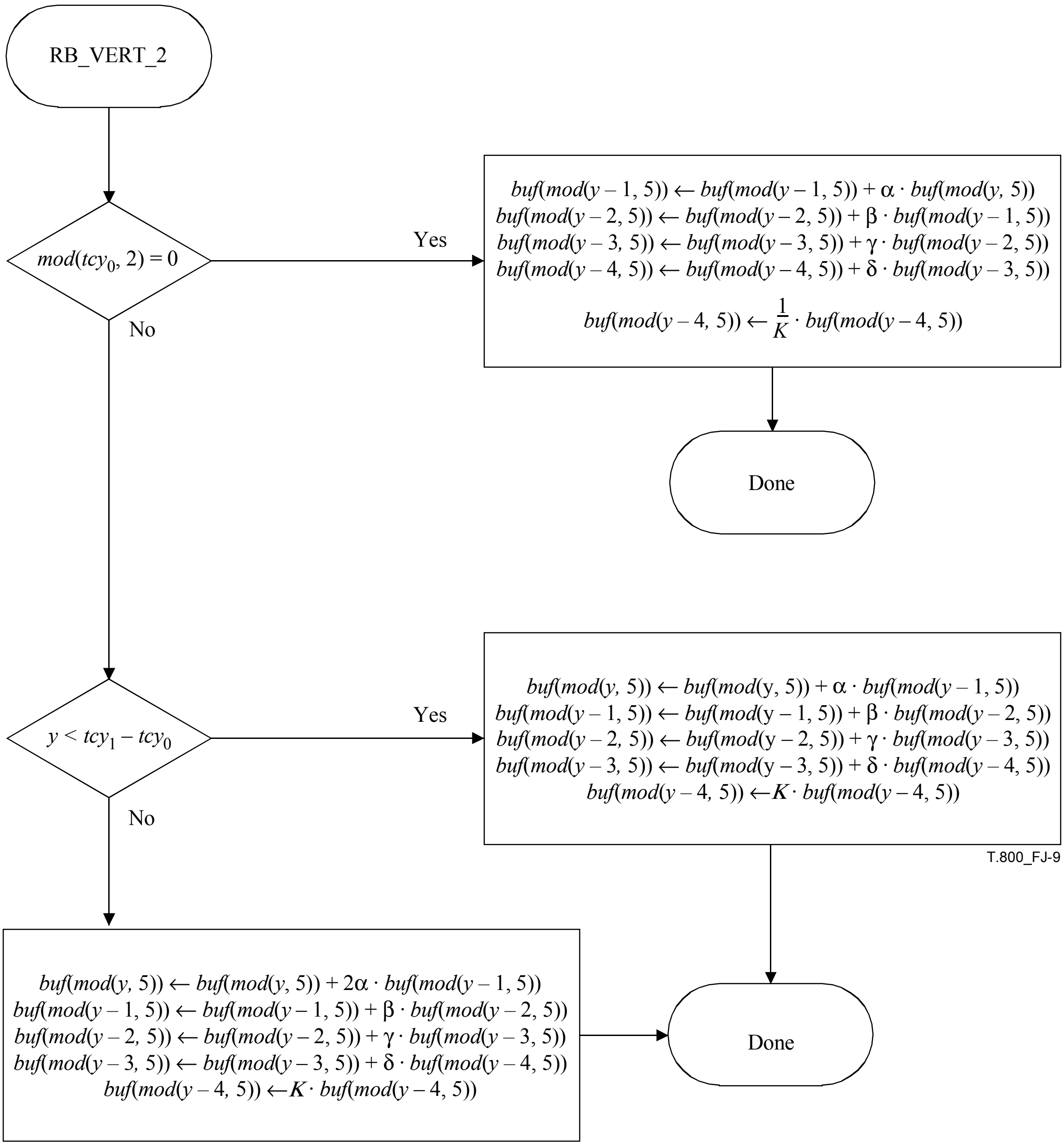


Figure J.9 – The RB_VERT_2 procedure

J.5.3.3 The END_1 procedure

The END_1 procedure is detailed in Figure J.10.

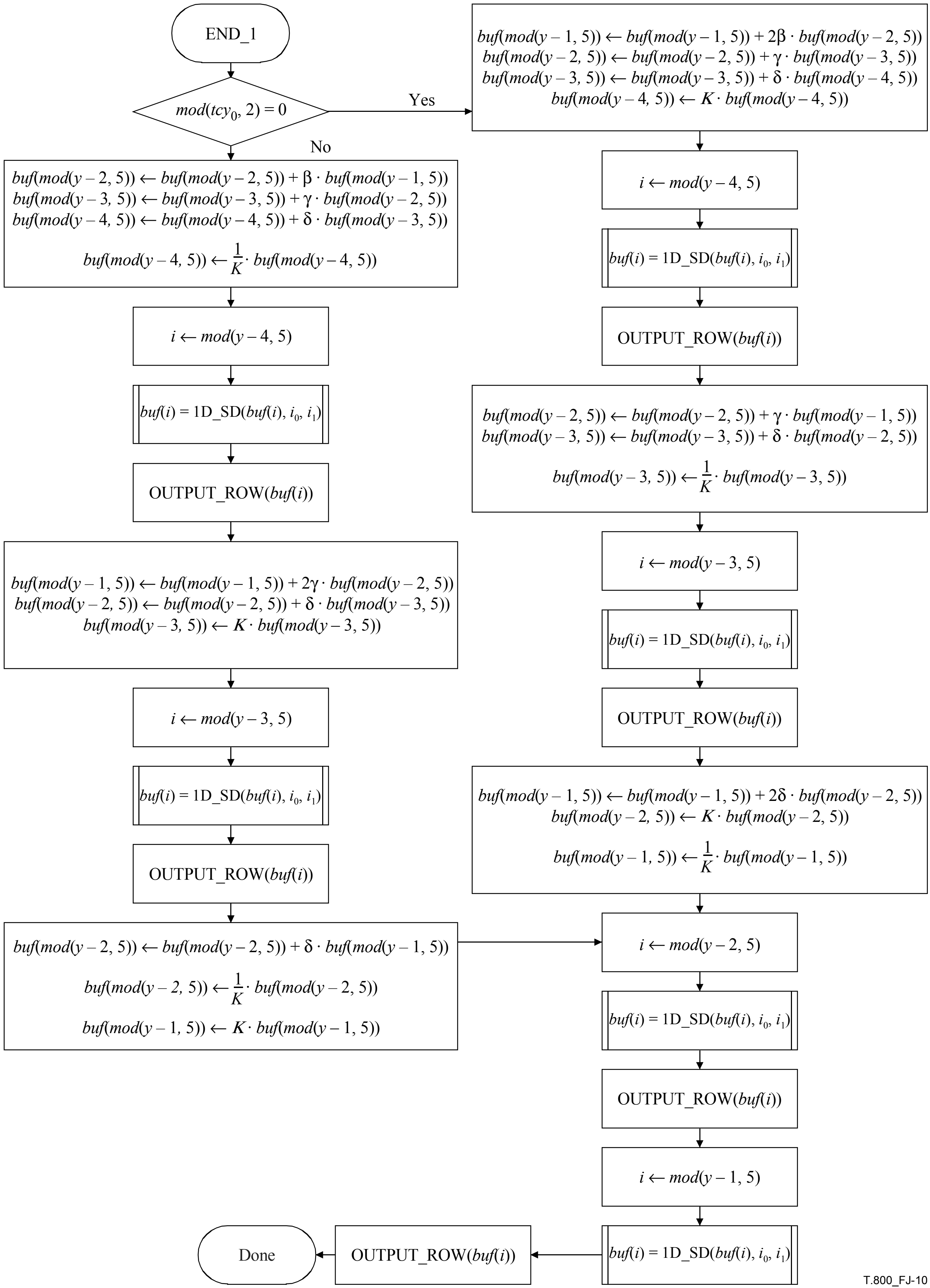


Figure J.10 – The END_1 procedure

J.5.3.4 The END_2 procedure

The END_2 procedure is detailed in Figure J.11

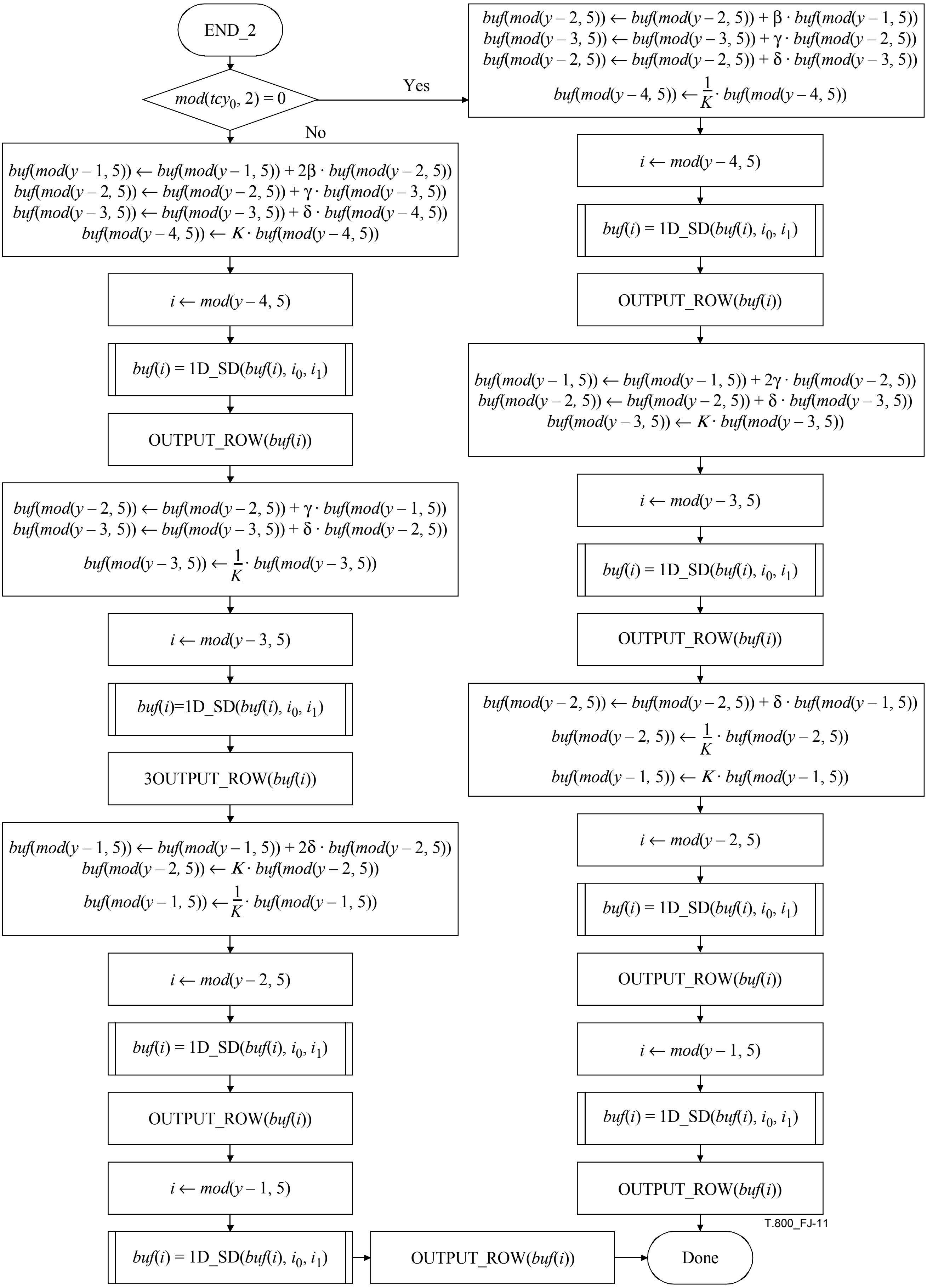


Figure J.11 – The END_2 procedure

J.5.4 OUTPUT_ROW procedure

This procedure returns a row *buf*(*i*) of transformed coefficients, which correspond either to the 1LL and 1HL sub-band or to the 1LH and 1HH sub-band. This row of transform coefficients can be either stored in an external memory or processed immediately.

J.6 Scan-based coding

Some applications use scanning sensors that create images (possibly unconstrained in length) row by row and have limited amounts of processing memory. These applications need full scan-based coding where only the minimum required number of bytes is retained in memory at any given time without significant loss in performance. Example implementations of such a scan-based coding systems have been demonstrated [39], [40]. The recommended procedure is outlined below.

The example rate control described in J.14 requires buffering of the entire compressed codestream at a bit rate higher than the interleaved final bit rate. Alternatively, a scan-based approach can be used where the row-based wavelet transformation (see J.5) is followed by a scan-based rate allocation and coding procedure to ensure that compressed wavelet coefficients are transmitted soon after they have been generated. For this purpose, a limited memory buffer (the scan buffer) is introduced after the wavelet transform. The discrete compressed data segments within it are called "scan elements". A scan element consists of a localized set of wavelet coefficients. It may be a tile or a precinct, and corresponds to a small number of rows in image space. The scan buffer may contain one or more scan elements.

The rate control algorithm is applied to the compressed data in the scan buffer and the first scan element is released to the bit stream. In case there is more than one scan element in the scan buffer, a sliding window rate control mechanism is implemented. This approach may give better compression results at the expense of a slight increase in complexity and memory requirements.

This scan-based approach does not affect the JPEG2000 decoding process.

J.7 Error resilience

This clause 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 Recommendation | International Standard.

The error resilience tools (see Table J.18) in this Recommendation | International Standard deal with channel errors using the following approaches: compressed data partitioning and resynchronization, error detection and concealment, and Quality of Service (QoS) transmission based on priority. Error resilience tools are described in each category.

Table J.18 – 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 segmentation symbols	Annex D
Packet level	short packet format packet with resynchronization marker (SOP)	Annex B

The entropy coding of the quantized coefficients is done within 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 within that code-block (see Annex D).

Termination of the arithmetic coder is allowed after every coding pass. Also, the contexts may be reset after each coding pass. This allows the arithmetic coder to continue to decode coding passes after errors (see D.4).

The optional arithmetic coding bypass style puts raw bits into the bit stream without arithmetic coding. This prevents the types of error propagation to which variable length coding is susceptible (see D.6).

Short packets are achieved by moving the packet headers to the PPM or PPT marker segments (see A.7.4 and 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.

A segmentation symbol is a special symbol. The correct decoding of this symbol confirms the correctness of the decoding of this bit-plane which allows error detection. (see D.5).

A packet with a resynchronization marker SOP (see A.8.1) allows spatial partitioning and resynchronization. This is placed in front of every packet in a tile with a sequence number starting at zero. It is incremented with each packet. Packet ordering is described in B.10.

J.8 Compatibility requirement with JFIF/SPIFF files

This clause only impacts extensions to the ITU-T Rec. T.81 | ISO/IEC 10918-1 specification and has no influence of any kind on this Recommendation | International Standard nor will it have in any of its extensions. There is no requirement to support this profile in a JPEG 2000 decoder.

J.8.1 Compatibility methodology

In order to avoid any type of modification in the file format described in the normative part of this Recommendation | International Standard, a new profile in the ITU-T Rec. T.81 | ISO/IEC 10918-1 is defined according to the extension methodologies described in ITU-T Rec. T.86 | ISO/IEC 10918-4. This profile will facilitate the transition from the use of ITU-T Rec. T.81 | ISO/IEC 10918-1 to this Recommendation | International Standard.

This new profile is available through the Registration Authority established for the purpose of extensions and is specified in ITU-T Rec. T.84 | ISO/IEC 10918-3 and ITU-T Rec. T.86 | ISO/IEC 10918-4 specifications. It will therefore appear on the JURA (JPEG Utilities Registration Authority) web site at http://jura.jpeg.org. The status of this new profile is the same as all other registered parameters; it is an extension to ITU-T Rec. T.81 | ISO/IEC 10918-1.

J.8.2 Compatibility design parameters

1) Transparent to a JFIF decoder (compliant with ITU-T Rec. T.81 | ISO/IEC 10918-1 and ITU-T Rec. T.83 | ISO/IEC 10918-2).
2) Transparent to a SPIFF decoder (compliant with ITU-T Rec. T.84 | ISO/IEC 10918-3).
3) A box basic structure of JP2/JPX.
4) No need to transcode DCT to wavelet codestream.
5) Preserves the integrity of, and access to, any IPR-related information.
6) Uses the ".jpg" or ".spf" extension according to its original file provenance.
7) Takes advantage of any registered parameter as described in ITU-T Rec. T.86 | ISO/IEC 10918-4.

J.9 Implementing the Restricted ICC method outside of a full ICC colour management engine

This annex describes the Restricted ICC method for specifying the colourspace of a JP2 file using ICC profiles based on version ICC.1:1998-09 of the ICC Profile Format Specification [41]. This annex is specifically targeted at developers who are not using a full ICC colour management engine and thus must extract the transformation parameters from the ICC profile and process the image using application specific code.

J.9.1 Extracting the colour transformation from an ICC profile

J.9.1.1 ICC profile format

An ICC profile uses a tagged data format to organize its information. It is described in clause 6 of the ICC Profile Format Specification. The format consists of a 128-byte header, a tag table, and tag data. Each tag is identified by a 32-bit signature which usually corresponds to four ASCII characters. The data for each tag is stored in a format which specifies the various data elements. Each format is identified by a data type signature, which is the first 32 bits of the tag data. To get the data for a tag, first locate the signature for that tag in the tag table, which specifies the position and size of the data for that tag, and then retrieve the data based on its position and size within the tag data. Once retrieved, the tag data type signature specifies how to interpret the tag data.

The important tags used in processing of an image through a Restricted ICC profile are summarized in Table J.19.

Table J.19 – Processing tags used by a Restricted ICC profile

Tag name	Tag signature	Tag data type	Tag data type signature
redTRCTag	'rTRC'	curveType	'curv'
greenTRCTag	'gTRC'	curveType	'curv'
blueTRCTag	'bTRC'	curveType	'curv'
redColorantTag	'rXYZ'	XYZType	'XYZ\040'
greenColorantTag	'gXYZ'	XYZType	'XYZ\040'
blueColorantTag	'bXYZ'	XYZType	'XYZ\040'
grayTRCTag	'kTRC'	curveType	'curv'

Note that an ICC profile, and thus a Restricted ICC profile, may contain other tags, such as *mediaWhitePoint*. While these tags are not used in the default processing path of a Restricted ICC profile as described in J.9.2, more complex rendering scenarios may take advantage of that information to provide a more accurate or optimized rendition of the image.

The ICC profile format, and thus a Restricted ICC profile, specifies a processing model which converts between device code values and the Profile Connection Space (PCS). This model consists of two parts, a set of three one-dimensional interpolation tables and a three-by-three matrix. The interpolation tables are formed from the redTRCTag, greenTRCTag, and blueTRCTag. The matrix is formed from the redColorantTag, greenColorantTag, and blueColorantTag. The basic processing model using these elements specified by Equation (4), clause 6.3.1.2 of the ICC Profile Format Specification and described in J.9.2. Note that the profile describes the device values to PCS conversion. The matrix and interpolation tables must be inverted to convert from the PCS to device values.

The complete specification of the format of an ICC profile (and thus a Restricted ICC profile) is contained in the ICC Profile Format Specification, version ICC.1:1998-09.

J.9.1.2 Interpolation tables

The interpolation tables use the curveType format. A tag of type curveType consists of a count followed by that number of unsigned 16-bit table entries.

If the count is 1, the single table entry is an encoded gamma value. In this case, the one-dimensional table is formed using the formula:

$$linear = \left(\frac{dVal}{dMax}\right)^{\frac{tVal}{256}} \times 65\,535 \qquad \text{(J-2)}$$

where *dVal* is the device component value *dMax* is the maximum device value, and *tVal* is the table entry value.

If the count is more than one, the entries are the values of an interpolation table. The first entry corresponds to a device value of 0 and the last entry corresponds to the maximum device value (e.g., 255 for 8-bit data, 65 535 for 16-bit data). The remaining entries are uniformly spaced between those two entries. For example, if you have 8-bit data and 6 entries, then the 4th entry corresponds to device value of 255 * ((4 – 1)/(6 – 1)) = 153. Note that 1 is subtracted from the position of the entry in the table to convert it to a zero-based index, and that 1 is subtracted from the number of entries to convert it to the number of intervals between entries.

To convert the interpolation table to a look-up table, the number of entries in the interpolation must be adjusted to match the number of possible device values. In the case of 8-bit data, there are 256 possible values. If the interpolation table has 6 entries, it would have to be expanded to 256 entries. There are several methods to do this and often linear interpolation is used. Using this method, entries between the supplied interpolation table entries are calculated by linearly interpolating between adjacent interpolation table entries using the device value as the interpolant. For example, consider a 6-entry interpolation table with values T3 in the 3rd entry and T4 in the 4th entry. Look-up table index 128 would have an interpolation table position of (128/255) * 5 = 2,5098. The look-up table value at index 128 is calculated by interpolating between the value at entry 3 and the value at entry 4 using an interpolant of 0,5098: value at index 128 = T3 + ((T4 – T3) * 0,5098).

Each entry in the table can be converted to floating point by dividing by 65 535.

When processing a pixel, each component of the pixel is applied to its corresponding look-up table. The floating point values obtained from the 3-element column vector seen in Equation (4) of clause 6.3.1.2 of the ICC Profile Format Specification.

J.9.1.3 Matrix

The matrix is formed from the values of the redTRCTag, greenTRCTag, and blueTRCTag. These tags use the XYZType, which contains three XYZNumberTypes. The first is the X component, the second is the Y component, and the third is the Z component. Each XYZNumberType contains a signed 32-bit integer which can be converted to floating point by dividing by 65 536. The XYZ values of each tag correspond to a row of the matrix, as shown in Equation (4) of clause 6.3.1.2 of the ICC Profile Format Specification. This matrix is multiplied by the column vector produced by the interpolation table to produce the XYZ PCS values.

J.9.1.4 Combining source and destination profiles

The profile embedded in the JP2 file describes how to convert the image data into the PCS. It is referred to as the "source" profile. Typically, the image data needs to be converted into the data for another device, such as the display. This device is referred to as the "destination" device and its profile is referred to as the destination profile. The conversion is done by combining the processing models of the source and destination profiles. Assuming that the destination device transformation is limited to a 3×3 matrix followed by a 1D table, the procedure would be:

1) Obtain the interpolation tables and matrix from the source profile.
2) Obtain the interpolation tables and matrix from the destination profile.
3) Invert the interpolation tables and matrix of the destination profile.
4) Combine the two matrices using matrix multiplication.

This produces an overall processing model which is a set of one-dimensional tables, a matrix, and a second set of one-dimensional tables. This can be used to convert the source image pixels into destination pixel images.

J.9.2 Colour processing equations for three-component RGB images

The goal of the Restricted ICC profile method is to restrict the set of all ICC profiles down to a set which can be described using a simple set of colour processing equations. The ICC specification defines this class of profile as Three-Color Matrix-Based Input Profiles (defined in clause 6.3.1.2 of the ICC Profile Format Specification) and Monochrome Input Profiles (defined in clause 6.3.1.1 of the ICC Profile Format Specification). Profiles in the Three-Color Matrix-Based Input Profile class can be described using the following equations:

$$\begin{aligned} linear_r &= redTRC\left[decompressed_r\right] \\ linear_g &= greenTRC\left[decompressed_g\right] \\ linear_b &= blueTRC\left[decompressed_b\right] \end{aligned} \tag{J-3}$$

$$\begin{bmatrix} connection_x \\ connection_y \\ connection_z \end{bmatrix} = \begin{bmatrix} redColorant_x & greenColorant_x & blueColorant_x \\ redColorant_y & greenColorant_y & blueColorant_y \\ redColorant_z & greenColorant_z & blueColorant_z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_y \\ linear_z \end{bmatrix} \tag{J-4}$$

where $decompressed_{rgb}$ is the original decompressed pixel and $connection_{xyz}$ is the pixel converted into the XYZ form of the Profile Connection Space (XYZ_{PCS}). In Equation (J-3), the three look-up tables are loaded from the Restricted ICC profile from the redTRCTag, greenTRCTag and blueTRCTag tags respectively, as defined in clauses 6.4.38, 6.4.18 and 6.4.4, respectively, in the ICC Profile Format Specification. The common data format of those tags is defined in clause 6.5.25 of the profile specification. In Equation (J-4), the rows of the matrix are loaded from the redColorantTag, greenColorantTag and blueColorantTag tags respectively, as defined in clauses 6.4.39, 6.4.19 and 6.4.5, respectively, in the ICC Profile Format Specification. The common data format of those tags is defined in clause 6.5.2 of the profile specification.

The Monochrome Input Profile class can be described with the following equation:

$$connection = grayTRC\left[device\right] \tag{J-5}$$

where device is the original decompressed pixel and connection is the achromatic channel of the profile connection space. In Equation (J-5), the look-up table is loaded from the Restricted ICC profile from the grayTRCTag, as specified in clause 6.3.17. The data format of that tag is defined in clause 6.5.2 of the profile specification.

J.9.3 Converting images to sRGB

One of the most common application scenarios will be the situation where an image specified using the Restricted ICC profile method must be converted to the sRGB colourspace for softcopy display (for example desktop editing and web browsers) [42].

This transformation is used in conjunction with the Restricted ICC method to create resulting sRGB values from original source colour values [47]. Where applicable, like transforms (1D look-up tables or matrices) may be combined to enhance processing performance. For this example, only the transformation from the Profile Connection Space (XYZ_{PCS}) will be shown. It may later be combined with the transforms in Equations (J-3) and (J-4).

To move colours encoded in the XYZ_{PCS} to colours encoded in the sRGB colour space, there are three pieces necessary to complete the transformation. These pieces are embodied in two 3×3 matrices and a per channel, linear to non-linear conversion equation which may be applied in practice through three one dimensional look-up tables.

The first matrix in the transformation is required to perform a chromatic adaptation transformation between the defined adaptive white point of the ICC Profile Connection Space (chromaticities of CIE D50) and the defined adaptive white point of sRGB (chromaticities of CIE D65). There are several different choices of transformation which can be used. For this example transformation, the Bradford chromatic adaptation transformation (BFD) will be used [43]. The Bradford transformation has been shown to produce accurate results and has been adopted as part of the CIE recommended colour appearance model (CIECAM97s) [44], [45]. The BFD transformation typically includes a linear and a non-linear portion. In the case of this example transform, the non-linear portion of the Bradford transformation has been left out to allow for simple 3×3 matrix processing. It has been shown that the Bradford transform's performance is still very good even with this omission [46].

The second matrix in the transformation is a primary transformation matrix required to move colours from the primaries of the XYZ_{PCS} to the ITU-R BT.709-2 primary set as defined in the sRGB standard, IEC/TC100/PT61966-2.1.

Separate, the transformation looks as follows with the primary transformation denoted by a PT and the Bradford chromatic adaptation matrix denoted by a BFD:

$$\begin{bmatrix} slinear_r \\ slinear_g \\ slinear_b \end{bmatrix} = \begin{bmatrix} 3{,}2406_{PT} & -1{,}5372_{PT} & -0{,}4986_{PT} \\ -0{,}9689_{PT} & 1{,}8758_{PT} & 0{,}0415_{PT} \\ 0{,}0557_{PT} & -0{,}2040_{PT} & 1{,}0570_{PT} \end{bmatrix} \begin{bmatrix} 0{,}9554_{BDF} & -0{,}0231_{BDF} & 0{,}0633_{BDF} \\ -0{,}0284_{BDF} & 1{,}0100_{BDF} & 0{,}0211_{BDF} \\ 0{,}0123_{BDF} & -0{,}0205_{BDF} & 1{,}3305_{BDF} \end{bmatrix} \begin{bmatrix} connection_x \\ connection_y \\ connection_z \end{bmatrix} \quad \text{(J-6)}$$

However, the matrices can be combined to form a single matrix as shown in the following equation:

$$\begin{bmatrix} slinear_r \\ slinear_g \\ slinear_b \end{bmatrix} = \begin{bmatrix} 3{,}1337 & -1{,}6173 & -0{,}4907 \\ -0{,}9785 & 1{,}9162 & 0{,}0334 \\ 0{,}0720 & -0{,}2290 & 1{,}4056 \end{bmatrix} \begin{bmatrix} connection_x \\ connection_y \\ connection_z \end{bmatrix} \quad \text{(J-7)}$$

It is then necessary to transformation the $slinear_{rgb}$ to non-linear sRGB values. For each red, green and blue channel of the $slinear_{rgb}$ value (referred to as $slinear_X$), the respective channel of the non-linear sRGB value (referred to as $sRGB_X$) is calculated using Equation (J-8).

$$sRGB_X = \begin{pmatrix} 12{,}92 \cdot slinear_X & slinear_X \le 0{,}0031308 \\ 1{,}055 \cdot slinear_X^{(1{,}0\,/\,2{,}4)} - 0{,}055 & slinear_X > 0{,}0031308 \end{pmatrix} \quad \text{(J-8)}$$

Note that the conversion from decompressed pixel to sRGB can be optimized by combining the colourant matrix described in Equation (J-4) with the XYZ to sRGB conversion matrix described in Equation (J-7) as follows:

$$\begin{bmatrix} slinear_r \\ slinear_g \\ slinear_b \end{bmatrix} = \begin{bmatrix} 3{,}1337 & -1{,}6173 & -0{,}4907 \\ -0{,}9785 & 1{,}9162 & 0{,}0334 \\ 0{,}0720 & -0{,}2290 & 1{,}4056 \end{bmatrix} \begin{bmatrix} redColorant_x & greenColorant_x & blueColorant_x \\ redColorant_y & greenColorant_y & blueColorant_y \\ redColorant_z & greenColorant_z & blueColorant_z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_z \end{bmatrix} \quad \text{(J-9)}$$

This optimization reduces the colourspace processing from decompressed pixel to sRGB to the application of a 1D look-up table, a single 3×3 matrix and another 1D look-up table.

The transforms shown above for sRGB can be generalized for use in converting to many other target colour spaces other than sRGB. In many cases, the steps taken will match exactly those needed for the conversion to sRGB. However, in other cases, fewer steps may be required such as when the adaptive white point of the target colour space matches that of the PCS XYZ thus removing the need for a chromatic adaptation transform. It is also possible that some cases may require additional steps to compensate for different factors such as viewing condition differences. The actual viewing condition transforms are beyond the scope of this annex, but have been covered in other publications [41], [42], [46], [48].

It should be noted that, depending on the storage colour space, there may be a loss of information involved with the conversion to sRGB, or any other limited colour gamut output colour space. For example, consider the case where the storage colour space is an extended colour gamut colour space. The conversion to sRGB will result in the clipping of colours that are outside the sRGB colour gamut. While this is a necessary step when displaying the image, colour transformations (such as a colour shift) may have been able to make use of that clipped data (by shifting into the sRGB gamut). As such, it is often preferable to perform most colour transformations on the original stored data before converting it to the sRGB colour space. Also, if the image were then to be printed on an output device that was capable of printing the colours that had been clipped, then it would be preferable to go back to the image in the storage colour space, rather than printing the sRGB image. As another example, consider the case where the storage colour space image is an extended dynamic range scene colour encoding. The conversion to sRGB will necessarily include a rendering step where highlight and/or shadow information is clipped to the dynamic range of the output display. (The rendering step is commonly implemented by applying a tonescale function as part of the TRCs shown in Equation (J-3). The information that is lost in the rendering step can not be used to modify the image at a later time. Similarly, the conversion to sRGB may introduce quantization errors to the image which would limit the quality of the image. In both of these examples, it may be desirable to retain the image in the storage colour space, and apply any image manipulations to the image there. Alternatively, the image could be converted to an intermediate large gamut colour encoding. The conversion to sRGB can be done for preview purposes only, or as a final step in the imaging chain.

J.9.4 Converting images to other colour spaces

Alternatively, it may be desirable in certain applications to convert images to other colour spaces besides sRGB for purposes of display on specific output devices, or manipulation in an application-specific colour space. For example, if it is desired to display the image on a CRT having known characteristics that are different than the reference sRGB display, then the matrix and nonlinearity specified in Equations (J-6) through (J-9) can be replaced with the corresponding matrix and nonlinearity for the particular CRT. Generally, a matrix related to the phosphor chromaticities and white point can be used to convert the PCS tristimulus values to the particular linear RGB values, and the nonlinearity can be used to relate the linear RGB values to the corresponding code values. Similarly, other additive RGB colour spaces, such as ROMM RGB can also be computed by substituting the appropriate matrix and nonlinearity [49].

In some cases, it may be desirable to convert images to other colour spaces that can not be described by a simple matrix/ nonlinearity transformation. This can be accomplished by replacing the transformation described in J.9.3 with the appropriate transformation to the desired colour space. In many cases, this can conveniently be accomplished by using an ICC profile for the desired colour space.

J.9.5 Input and output ranges and quantization

The input code values to the look-up tables in Equation (J-3) (redTRC, greenTRC and blueTRC) shall be integers of the same precision as the decompressed code values, and indexed such that TRC[i] produces the correct linear intensity value for an input code value of *i*. Input code values that are larger than the number of elements of the look-up table – 1 should be clipped to the number of elements of the look-up table – 1.

The output pixel from Equation (J-3) shall be real linear intensity values nominally in the range (0.0, 1.0).

The input to the colourant matrix in Equation (J-4) shall also be real linear intensity values in the range (0.0, 1.0). The output of that equation (the XYZ_{PCS} values) is scaled such that the Y value will be in the range (0.0, 1.0). Neutral values in the image should map to XYZ values having the chromaticity of the PCS whitepoint (this implies that X/Y = 0.9642, and Z/Y = 0.8250). If the application is converting the input code values to the sRGB colourspace, this output range allows direct concatenation of the matrices as in Equation (J-8).

The ranges and quantization of the XYZ_{PCS} to sRGB transformation are similar. The inputs and outputs of Equation (J-6), and thus the inputs to Equation (J-8), are also real values in the range (0.0, 1.0).

The outputs of Equation (J-8) are values in the range (0.0, 1.0). However, those values will generally be scaled by 255 to produce 8-bit sRGB values. This is highly application dependent and depends on what, if any, additional processing will be performed. However, it is strongly suggested that any colour processing be performed on the source image data (decompressed$_r$, decompressed$_g$, decompressed$_b$) before it is converted to sRGB, as the possibility of significantly decreased quantization exists.

J.9.6 Taking advantage of multiple colourspace specifications

The JP2 format allows for a file to specify multiple methods to interpret the colourspace of an image. For example, one application may write images in which the pixel values have already been converted to the signals necessary for driving a particular output device. In that situation, it is useful for the application to provide a simple mechanism for the device to determine that additional colour processing is not required. This can be accomplished by specifying the name of the device colourspace using the Enumerated Colourspace method in one Colour specification box in the file.

However, other applications, such as web browsers, must convert the image to signals suitable for display on other devices; it is very likely that those applications will not know the definition of this vendor-specific colourspace. It is thus very useful for the original file writer to write a second Colour specification box in the file that uses the Restricted ICC profile method or the Generic ICC profile method. By providing a secondary mechanism, the number of applications that have the ability to properly interpret the colourspace of the image is dramatically increased.

Note that the method for choosing from the multiple colour specification methods defined in a single file is not specified by this Recommendation | International Standard. Each application should select the method which best meets the requirements of that particular application.

J.10 An example of the interpretation of multiple components

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. Bypassing 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.

J.11 An example of decoding showing intermediate steps

Consider the following compressed bit stream where the offset from the beginning of the file is given in octal on the left, and the values in the file are given in Hexidecimal.

```
00000  FF4F  FF51  0029  0000  0000  0001  0000  0009
00020  0000  0000  0000  0000  0000  0001  0000  0009
00040  0000  0000  0000  0000  0001  0701  01FF  5C00
00060  0740  4048  4850  FF52  000C  0000  0001  0001
00100  0404  0001  FF90  000A  0000  0000  001E  0001
00120  FF93  C7d4  0C01  8F0D  C875  5DC0  7C21  800F
00140  B176  FFD9
```

The various parts of this bit stream can be decoded as follows.

J.11.1 Main header

The main header starts at byte 0 as indicated by the SOC marker and ends before byte 0104 (octal), which is known because of the SOT marker.

```
00000  FF4F                 SOC marker
00002  FF51                 SIZ marker
00004  0029                 Lsiz SIZ marker length
00006  0000                 Rsiz
00010  0000  0001           Xsiz
00014  0000  0009           Ysiz
00020  0000  0000           XOsiz
00024  0000  0000           YOsiz
00030  0000  0001           XTsiz
00034  0000  0009           YTsiz
00040  0000  0000           XTOsiz
```

```
00044  0000  0000          YTOsiz
00050  0001                Csiz
00052  07                  Ssiz
00053  01                  XRsiz
00054  01                  YRsiz
```

Thus the "image" is one component, with 8 bits/sample unsigned, 1 sample horizontally, and 9 samples vertically, and all samples are in a single tile.

```
00055  FF5C                QCD marker
00057  0007                Lqcd QCD marker length
00061  40                  Sqcd
00062  4048  4850          SPqcd
```

There are 2 guard bits, no quantization is done (other than possible truncation), and the quantizer step size exponents ε_b are {8,9,9, 10}.

```
00066  FF52                COD marker
00070  000C                Lcod COD marker length
00072  00                  Scod (PPx = PPy = 15, No SOP, No EPH)
00073  00                  Progression order
00074  0001                Number of layers
00076  00                  Multiple component transform
00077  01                  Number of decomposition levels
00100  04                  Code-block width exponent offset value
00101  04                  Code-block height exponent offset value
00102  00                  Style of the code-block coding passes
00103  01                  Transform
```

No precincts are used. There is one level of wavelet transform. Progression is layer-resolution level-component-position, but there is only one layer. Code-blocks are 64×64 samples (note the size is 2^6 while the value in the bit stream is 4). There is no selective arithmetic coding bypass, no reset of context probabilities or termination at each coding pass, no vertical stripe causal contexts, no predictable termination, and no segmentation symbols. The 5-3 reversible filter is used.

J.11.2 Tile-part header

The first and only tile-part header begins at byte 0104 octal with the SOT marker and ends at byte 0120 octal with the SOD marker.

```
00104  FF90                SOT marker
00106  000A                Lsot SOT marker length
00110  0000                Isot
00112  0000  001E          Psot
00116  00                  TPsot
00117  01                  TNsot
```

This is tile number 0. The length of the tile-part is 30 bytes. Thus the next tile-part or the end of codestream is at 0104 + 036 = 0142. This is tile-part 0. There is only one tile-part for this tile.

```
00120  FF93                SOD marker
```

There were no COD, or QCD or comment markers in this tile-part header. Thus all coding parameters are determined from the main header. The next 16 bytes are compressed data (30-byte length – 14 bytes of marker segments).

Compressed data (packet headers and packet bodies)

```
00122  C7D4  0C01  8F0D  C875  5DC0  7C21  800F
00140  B176
```

End of Image

```
00142  FFD9                    EOC marker
```

J.11.3 Packet headers

Because the image is 1 × 9, and there is one level of transform, (and the code-blocks, precincts, and tiles are too large to have an effect), there will be 5 low-pass wavelet coefficients, and 4 horizontal low-pass vertical high-pass coefficients. The compressed data begins with a packet header which is decoded as shown in Table J.20.

Table J.20 – Decoding first packet header

Codestream bytes	Bits used	Comments
0xC7	1	Nonzero length packet
	1	Only code-block is included
	0001	3 zero bit-planes[a)]
0xD4	11 1101010	16 coding passes for this code-block[b)]
	0	*LBlock* remains 3
0x0C	0000110	6 bytes of compressed data[c)]
	0	unused padding bit

a) Maximum bit-planes from Equation (E-2) is 9, 3 of them are zero, so first bit will be decoded into bit-plane 6.

b) Number of bits to read depends on compressed data; see Table B.2.

c) Number of bits to read is *LBlock* plus floor of base 2 log of number of passes: $3 + \lfloor \log 16 \rfloor = 7$.

Decoding the first packet header requires 3 bytes and indicates 6 bytes of arithmetic coded compressed data are used for the only code-block in this packet. Thus the next packet header begins at offset 0134. This packet header is decoded in Table J.21.

Table J.21 – Decoding second packet header

Codestream bytes	Bits used	Comment
0xC0	1	Non-zero length packet
	1	Only code-block is included
0x7C	000000 01	7 zero bit-planes[a)]
0x21	111100 001	7 coding passes for this code-block
	0	*LBlock* remains 3
0x80	0001 1	3 bytes of compressed data[b)]
	0000000	unused padding bits

a) Maximum bit-planes is 10, 7 of them are zero, so first bit will be decoded into bit-plane 3.

b) Number of bits to read is $3 + \lfloor \log 7 \rfloor = 5$.

J.11.4 Arithmetic-coded compressed data

The six bytes of compressed data for the first code-block (from the first packet) can be decoded as shown in Table J.22. The first item is the context label from Annex D (which could be completely different for each implementation). The second item is the type of context. Finally the bit returned from the arithmetic coder is listed. These bits are used to determine the low-pass horizontal low-pass vertical coefficients. The bytes provided to the arithmetic coder are those beginning at offset 0125.

```
0000125   01   8F0D   C875   5D
```

Table J.22 – Arithmetic decode of first code-block

CTX	Context Type	Bit	Comment
17	C4(ZERO_RUN)	1	No zero run.
18 18	C5(UNIFORM) C5(UNIFORM)	1 1	First nonzero coefficient is the fourth (numbered from 1).
9	C2(SIGN)	1	Negative.
3	C1 (NEW_SIGNIFICANT)	0	Fifth coefficient is not significant.
3	C1 (NEW_SIGNIFICANT)	1	Third coefficient is significant (first coefficient which is in the significance pass).
10	C2 (SIGN)	0	Negative (XOR bit is 1).
3	C1 (NEW_SIGNIFICANT)	1	Fifth coefficient significant in this coding pass.
10	C2 (SIGN)	0	Negative (XOR bit is 1)
15	C3 (REFINE)	0	Next bit of fourth coefficient is 0.
0	C1 (NEW_SIGNIFICANT)	1	First coefficient is significant.
9	C2 (SIGN)	1	Negative.
4	C1 (NEW_SIGNIFICANT)	1	Second coefficient is significant.
10	C2 (SIGN)	0	Negative.
15 15 15 16 15	C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE)	1 0 1 0 0	All coefficients are in the refinement pass. Decoded bit is the next bit of the coefficient in order from first to fifth.
16 16 16 16 16	C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE)	0 1 1 0 0	Next bit-plane.
16 16 16 16 16	C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE)	1 1 1 0 1	Next bit-plane.
16 16 16 16 16	C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE) C3 (REFINE)	0 0 0 0 1	Last bit-plane.

Thus the decoded coefficients are:

```
-26, -22, -30, -32, -19
```

The compressed data for the only code-block in the second packet, representing the vertical high pass horizontal lowpass sub-band begins at offset 0137 octal.

```
0000137 0F B176
```

The decoding process is described in Table J.23.

Table J.23 – Arithmetic decode of second code-block

CTX	Context Type	Bit	Comment
17	C4 (ZERO_RUN)	1	Not a zero run.
18 18	C5 (UNIFORM) C5 (UNIFORM)	0 1	First nonzero coefficient is 2nd.
9	C2 (SIGN)	0	Positive.
3 0	C1 (NEW_SIGNIFICANT) C1 (NEW_SIGNIFICANT)	0 0	3rd and 4th coefficients in clean-up pass.
3 3	C1 (NEW_SIGNIFICANT) C1 (NEW_SIGNIFICANT)	0 0	1st and 3rd coefficients in significance pass.
14	C3 (REFINE)	0	2nd coefficient.
0	C1 (NEW_SIGNIFICANT)	0	4th coefficient in clean-up pass.
3	C1 (NEW_SIGNIFICANT)	1	1st coefficient in significance pass.
10	C2 (SIGN)	0	Positive.
3	C1 (NEW_SIGNIFICANT)	1	3rd coefficient in significance pass.
10	C2 (SIGN)	0	Positive.
3	C1 (NEW_SIGNIFICANT)	0	4th coefficient in significance pass.
16	C3 (REFINE)	1	2nd coefficient in refinement pass.

The decoded vertical high pass horizontal low pass coefficients are:

```
1, 5, 1, 0
```

J.11.5 Wavelet and level shift

After the inverse 5-3 reversible filter and level shifting, the component samples in decimal are:

```
101, 103, 104, 105, 96, 97, 96, 102, 109
```

J.12 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 transformation 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 [34] and [35] 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 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 details of the image are revealed. The image is usually examined at a closer distance, or is even magnified for close examination, which is equivalent to decreasing 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 visual progressive coding. It is clear that fixed visual weighting can be viewed as a special case of visual progressive coding.

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 subband. The set of CSF weights can be incorporated in one of the following two ways.

J.12.1 Modify quantization step size

At the encoder, the quantization step size q_i of the transformation coefficients of the *i*th sub-band is adjusted to be inversely 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 do not need to be transmitted to the decoder. The information is included in the quantization step sizes, which are explicitly transmitted for each sub-band.

This approach needs to explicitly specify the quantizer. Therefore, it may not be very suitable for embedded coding, especially for embedded coding from lossy all the way to lossless.

J.12.2 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 does not need to be transmitted explicitly. This approach is recommended since it produces similar results in J.12.1 and is compatible with lossless compression. This approach affects only the compressor and it is compatible with all quantization strategies, including implicit quantization.

J.12.3 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:

$$\begin{gathered}\text{Weighting set } 0 : r(0), \text{ with } W(0) = \{w_0(0), w_1(0), \ldots, w_n(0)\}; \\ \text{Weighting set } 1 : r(1), \text{ with } W(1) = \{w_0(1), w_1(1), \ldots, w_n(1)\}; \\ \ldots \\ \text{Weighting set } m : r(m), \text{ with } W(m) = \{w_0(m), w_1(m), \ldots, w_n(m)\},\end{gathered} \tag{J-10}$$

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 over the 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 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). 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, progressively changing visual weights effectively controls the embedding order of code-block sub-bit-planes on the fly.

J.12.4 Recommended frequency weighting tables

Table J.24 specifies three sets of CSF weights which were designed for the luminance component based on the CSF value at the mid-frequency of each sub-band. The viewing distance is supposed to be 1000, 2000, and 4000 samples (e.g., corresponding to 10 inches for 100 dpi, 200 dpi, and 400 dpi print or display), respectively. Note that the tables are intended for a 5-level wavelet decomposition.

Table J.24 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. (HL, LH, HH) denotes the three frequency orientations within each sub-band.

Table J.24 – Recommended frequency weighting

Level	Viewing distance 1000			Viewing distance 2000			Viewing distance 4000		
	HL	LH	HH	HL	LH	HH	HL	LH	HH
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	0,731 668
3	1	1	1	1	1	0,727 203	0,564 344	0,564 344	0,285 968
4	1	1	0,727 172	0,560 841	0,560 841	0,284 193	0,179 609	0,179 609	0,043 903
5	0,560 805	0,560 805	0,284 173	0,178 494	0,178 494	0,043 631	0,014 774	0,014 774	0,000 573

For colour images, the frequency weighting tables of the Y, Cr, and Cb components should differ in order to take advantage of the properties of the human visual system. For example, it is usually desirable to emphasize the luminance component more than the chrominance components. Table J.25 specifies three sets of CSF weights for luminance and chrominance components.

Table J.25 – Recommended frequency weighting for multiple component (colour) images

Comp	Lev	Viewing distance 1000			Viewing distance 1700			Viewing distance 3000		
		HL	LH	HH	HL	LH	HH	HL	LH	HH
Y (Y0)	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	0,921 045	0,921 045	0,848 324
	4	0,998 276	0,998 276	0,996 555	0,861 593	0,861 593	0,742 342	0,410 628	0,410 628	0,182 760
	5	0,756 353	0,756 353	0,573 057	0,307 191	0,307 191	0,108 920	0,038 487	0,038 487	0,003 075
Cb (Y1)	1	0,883 196	0,883 196	0,833 582	0,818 766	0,818 766	0,745 875	0,717 086	0,717 086	0,613 777
	2	0,793 487	0,793 487	0,712 295	0,689 404	0,689 404	0,579 220	0,539 437	0,539 437	0,403 353
	3	0,650 482	0,650 482	0,531 700	0,501 652	0,501 652	0,362 279	0,319 773	0,319 773	0,185 609
	4	0,450 739	0,450 739	0,309 177	0,280 068	0,280 068	0,152 290	0,124 021	0,124 021	0,044 711
	5	0,230 503	0,230 503	0,113 786	0,097 816	0,097 816	0,031 179	0,023 308	0,023 308	0,003 413
Cr (Y2)	1	0,910 877	0,910 877	0,872 378	0,860 885	0,860 885	0,803 172	0,780 091	0,780 091	0,695 128
	2	0,841 032	0,841 032	0,776 180	0,757 626	0,757 626	0,665 951	0,631 632	0,631 632	0,509 729
	3	0,725 657	0,725 657	0,625 103	0,598 537	0,598 537	0,470 893	0,428 659	0,428 659	0,287 593
	4	0,552 901	0,552 901	0,418 938	0,388 492	0,388 492	0,248 566	0,211 871	0,211 871	0,100 658
	5	0,336 166	0,336 166	0,200 507	0,177 435	0,177 435	0,077 130	0,060 277	0,060 277	0,014 977

J.13 Encoder sub-sampling of components

It has become common practice in some compression applications to utilize component sub-sampling in conjunction with certain decorrelating transforms. A typical example is the use of an RGB to YCrCb decorrelation transformation followed by sub-sampling of the chrominance (Cr, Cb) components. While this is an effective way to reduce the amount of image data to encode for DCT-based compression algorithms (ITU-T Rec. T.81 | ISO/IEC 10918-1), it is not recommended for use in this Recommendation | International Standard.

The multi-resolution nature of the wavelet transformation described in this Recommendation | International Standard may be used to achieve the same effect as that obtained from component sub-sampling. For example, if the 1HL, 1LH, and 1HH sub-bands of a component's wavelet decomposition are discarded and all other sub-bands retained, a 2:1 sub-sampling has been achieved in the horizontal and vertical dimensions of the component. This technique provides the same benefits as explicitly sub-sampling the component prior to any wavelet transform.

Furthermore, it frequently proves to be beneficial in terms of image quality to retain a few of the wavelet coefficients in the 1HL, 1LH, 1HH subbands, while still discarding the vast majority. In such cases the number of coefficients is still approximately reduced 2:1, but the resultant decoded imagery will exhibit better quality with fewer compression artifacts. Using a sub-sampling technique denies encoders from making such choices and can impair decoded image quality.

J.14 Rate control

Rate control is useful for meeting a particular target bit-rate or transmission time. Rate control assures that the desired number of bytes is used by the codestream while assuring the highest image quality possible.

J.14.1 Introduction to key concepts for rate control

Divide each subband into code-blocks of samples which are coded independently. Since every code-block is coded completely independently using exactly the same algorithm in every sub-band, the association between sub-bands and code-blocks can be ignored for the moment and let $\{B_i\}_{i=1,2,...}$ denote the set of all code-blocks which represent the image. For each code-block, B_i, a separate bit-stream is generated without utilizing any information from any of the other code-blocks. Moreover, the bit-stream has the property that it can be truncated to a variety of discrete lengths $R_i^1, R_i^2, R_i^3, \ldots$, and the distortion incurred when reconstructing each of these truncated subsets is estimated and denoted by $D_i^1, D_i^2, D_i^3, \ldots$ The mean squared error distortion metric is often used, but this is not necessary. During the encoding process, the lengths, R_i^n, and the distortions, D_i^n, are computed and temporarily stored in a compact form with the compressed bit-stream itself.

Once the entire image has been compressed, a post-processing operation passes over all the compressed code-blocks and determines the extent to which each code-block's embedded bit-stream should be truncated in order to achieve a particular target bit-rate, distortion bound or other quality metric. More generally, the final bit-stream is composed from a collection of so-called "layers," where each layer has an interpretation in terms of overall image quality. The first, lowest quality layer, is formed from the optimally truncated code-block bit-streams in the manner described above. Each subsequent layer is formed by optimally truncating the code-block bit-streams to achieve successively higher target bit-rates, distortion bounds or other quality metrics, as appropriate, and including the additional code words required to augment the information represented in previous layers to the new truncation points. These layered bit-stream concepts are discussed further in J.14.2.

J.14.2 Layered Bit-Stream Abstraction

An important aspect is the manner by which the encoder forms a final bit-stream from the independent embedded bit-streams generated for every code-block. The bit-stream formation problem is very much simplified when the coder operates on entire sub-bands at a time, since the additional spatial organization imposed by independent code-blocks does not exist.

Basically, the bit-stream is organized as a succession of layers, where each layer contains the additional contributions from each code-block (some contributions may be empty), as illustrated in Figure J.12. The code-block truncation points associated with each layer are optimal in the rate-distortion sense, which means that the bit stream obtained by discarding a whole number of least important layers will always be rate-distortion optimal. If the bit stream is truncated part way through a layer then it will not be strictly optimal, but the departure from optimally can be small if the number of layers is large. As the number of layers is increased so that the number of code bytes in each layer is decreased, the rate-distortion slopes associated with all code-block truncation points in the layer will become increasingly similar; however, the number of code-blocks which do not contribute to the layer will also increase so that the overhead associated with identifying the code-blocks which do contribute to the layer will increase. In practice, it is found that optimal compression performance for SNR progressive applications is achieved when the number of layers is approximately twice as large as the number of sub-bit-plane passes made by the entropy coder. The boundaries of the sub-bit-plane passes are also the truncation points for each code-block's embedded bit-stream. Consequently, on average each layer contains contributions from approximately half the code-blocks so that the cost of identifying whether or not a code-block contributes to any given layer (about 2 bits per code-block) is much less than the cost of identifying a strict order on the code-block contributions. Moreover, the relative contribution of this overhead to the overall bit-rate is independent of the size of the image.

Figure J.12 is an illustration of code-block contributions to bit-stream layers. Only five layers are shown with seven code-blocks, for simplicity. Notice that not all code-blocks need contribute to every layer and that the number of bytes contributed by code-blocks to any given layer is generally highly variable. Notice also that the code-block coding operation proceeds vertically through each code-block independently, whereas the layered bit-stream organization is horizontal, distributing the coding passes of the code-block to the various layers.

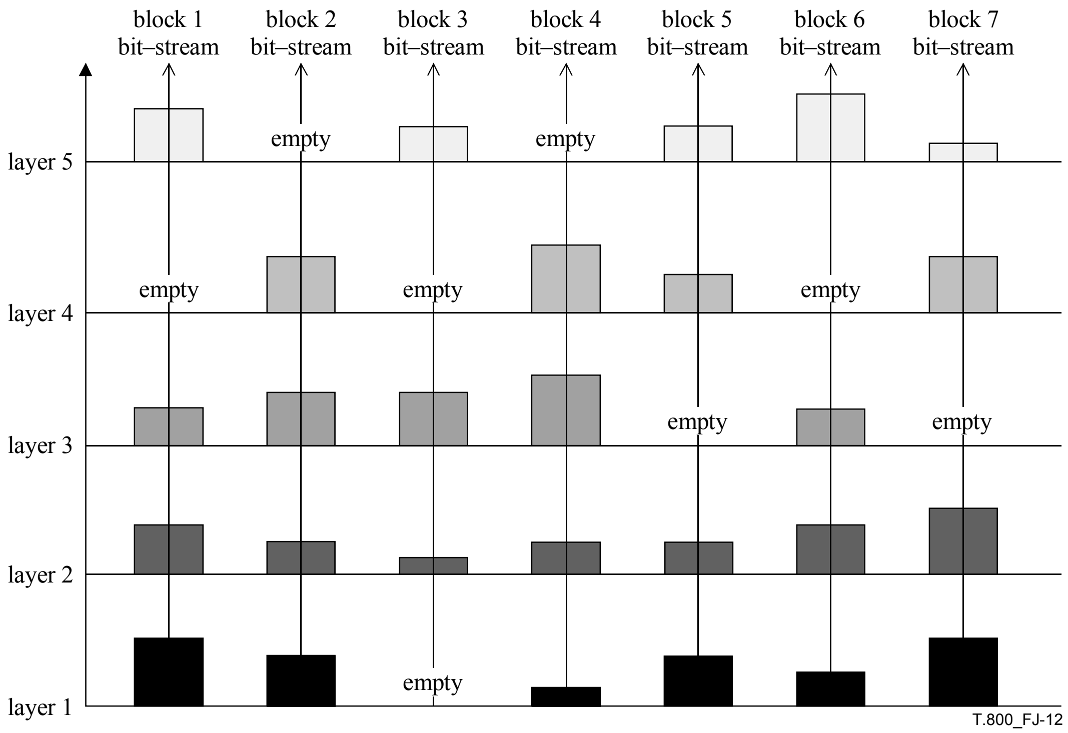


Figure J.12 – Illustration of code-block contributions to bit-stream layers

J.14.3 Rate-distortion optimization

The rate-distortion algorithm described here is justified only provided the distortion measure adopted for the code-blocks is additive. That is, the distortion, D, in the final reconstructed image should satisfy:

$$D = \sum_i D_i^{n_i} \tag{J-11}$$

where n_i is the truncation point for code-block B_i. Subject to suitable normalization, this additive property is satisfied by Mean Squared Error (MSE) and Weighted MSE (e.g., visually weighted MSE), provided the wavelet transformation is orthogonal. Additivity also holds if the quantization errors for individual sample values are uncorrelated, regardless of whether or not the transformation is orthogonal. In practice, the transformation is usually only approximately orthogonal and the quantization errors are not completely uncorrelated, so even squared error metrics are only approximately additive, but this is usually good enough. Let R denote the number of code bytes associated with some layer in the bit-stream (and all preceding layers). Then, for some set of truncation points, n_i:

$$R = \sum_i R_i^{n_i} \tag{J-12}$$

The need is to find the set of n_i values which minimizes D subject to the constraint $R \le R_{max}$. Constrained optimization problem by the method of Lagrange multipliers is a well-known solution to this problem. Specifically, the problem is equivalent to minimizing:

$$\sum \left(R_i^{n_i} - \lambda D_i^{n_i} \right) \tag{J-13}$$

where the value of λ must be adjusted until the rate yielded by the truncation points which minimize Equation (J-13) satisfies $R = R_{max}$. There is no simple algorithm which can yield a globally optimal set of truncation points in general. However, any set of truncation points, n_i, which minimizes Equation (J-13) for some λ is guaranteed to be optimal in the sense that minimum distortion is achieved at the corresponding bit-rate. If the largest value of λ is found such that the set of truncation points, n_i, obtained by minimizing Equation (J-13), yields a rate $R \le R_{max}$, then it is not possible to find any set of truncation points which will yield a smaller overall distortion and a rate which is less than or equal to R.

In practice, it is found that it is usually possible to find values of λ, such that R is very close to R_{max} (almost always within 100 bytes), so that any residual sub-optimally is of little concern.

Returning now to the problem of minimizing the expression in Equation (J-13), it is a separate optimization problem for each individual code-block. Specifically, for each code-block, B_i, the truncation point, n_i, needs to be found which minimizes $\left(R_i^{n_i} + \lambda D_i^{n_i}\right)$. A simple algorithm to do this is as follows:

Set $n_i = 0$ (i.e., no information included for the code-block)

For $k = 1,2,3,...$

Set $\Delta R_i^k = R_i^k - R_i^{n_i}$ and $\Delta D_i^k = D_i^k - D_i^{n_i}$

If $\left(\Delta D_i^k / \Delta R_i^k\right) > \lambda^{-1}$ then set $n_i = k$

Since this algorithm might need to be executed for many different values of λ, it makes sense to first identify the subset, N_i, of thresholds such that the rate-distortion slope values, $S_i^k = \Delta D_i^k / \Delta R_i^k$, are monotonically decreasing with k, for all k in N_i. Specifically, a suitable algorithm for determining N_i is as follows:

1) Set $N_i = \{n\}$, i.e., the set of all truncation points.
2) Set $p = 0$
3) For $k = 1, 2, 3, 4,...$

   If k belongs to N_i

      Set $\Delta R_i^k = R_i^k - R_i^p$ and $\Delta D_i^k = D_i^p - D_i^k$

      Set $S_i^k = \Delta D_i^k / \Delta R_i^k$

   If $p \neq 0$ and $S_i^k > S_i^p$ then remove p from N_i and go to step (2)

   Otherwise, set $p = k$

Once this information has been pre-computed, the optimization task for any given λ is simply to set P equal to the largest k in N_i such that $S_i^k > \lambda^{-1}$. Clearly, λ may be interpreted as a quality parameter, since larger values of λ, correspond to less severe truncation of the code-block bit streams; its inverse may be identified as a rate-distortion slope threshold.

The set N_i and the slopes S_i^k are computed immediately after code-block B_i is coded, and enough information to later determine the truncation points which belong to N_i and the corresponding R_i^k and S_i^k values during the rate-distortion optimization phase is stored. This information is generally smaller than the bit-stream itself which is stored for the code-block.

J.14.4 Efficient distortion estimation for R-D optimal truncation

The candidate truncation points for the embedded bit-stream representing each code-block correspond to the conclusion of each coding pass. During compression, the number of bytes, R_i^n, required to represent all coded symbols up to each truncation point, n, as well as the distortion, D_i^k, incurred by truncating the bit stream at each point, n, must be assessed. Actually, distortion estimation is not strictly necessary to generate a legal decompressible bit-stream, but it is important to the success of the rate-distortion optimization algorithm described in J.14.3.

J.14.4.1 Considerations for non-reversible transformations

The rate-distortion optimization algorithm described in J.14.3 depends only on the amount by which each coding pass reduces the distortion. Specifically, if D_i^0 denotes the distortion incurred by skipping the code-block altogether (i.e., setting all samples to zero), then only the differences, $D_i^n - D_i^{n-1}$, need to be computed for $n = 1, 2, 3,...$ It turns out that this computation can be performed with the aid of two small look-up tables which do not depend upon the coding pass, bit-plane or sub-band involved. To see this, let $\omega_i \Delta_i^2$ denote the contribution to distortion in the reconstructed image which would result from an error of exactly one step size in a single sample from code-block B_i.

Here ω_i is a positive weight, which is computed from the L2 norm of the relevant sub-band's wavelet synthesis waveform and may, additionally be modified to reflect visual weighting or other criteria. Now define:

$$v_i^p[m,n] = 2^{-p} v_i[m,n] - 2\left\lfloor \frac{2^{-p} v_i[m,n]}{2} \right\rfloor \quad \text{(J-14)}$$

Thus, $v_i^p[m,n]$ holds the normalized difference between the magnitude of sample $s_i[m, n]$ and the largest quantization threshold in the previous bit-plane which was not larger than the magnitude. It is easy to verify that $0 \le v_i^p[m,n] \le 2$. Although $s_i[m, n]$ is actually a quantized integer quantity, we will allow for the fact that the quantizer can supply fractional bits for $s_i[m, n]$ and hence $v_i[m, n]$, which can be used in Equation (J-14) to produce accurate estimates of the distortion associated with coding passes in the less significant bit-planes. Now when a single sample first becomes significant in a given bit-plane, p, we must have $v_i[m,n] \ge 2^p$ and hence $v_i^p[m,n] \ge 1$ and the reduction in distortion may be expressed as:

$$2^{2p}\omega_i \Delta_i^2 \left[v_i^p[m,n]^2 - \left(v_i^p[m,n] - 1{,}5\right)^2\right] = 2^{2p}\omega_i \Delta_i^2 f_s\left(v_i^p[m,n]\right) \quad \text{(J-15)}$$

provided the representation levels used during inverse quantization are midway between the quantization thresholds. Also, the reduction in distortion which may be attributed to magnitude refinement of a sample in bit-plane p may be expressed as:

$$2^{2p}\omega_i \Delta_i^2 \left[\left(v_i^p[m,n] - 1\right)^2 - \left(v_i^p[m,n] - 0{,}5 - v\right)^2\right] = 2^{2p}\omega_i \Delta_i^2 f_m\left(v_i^p[m,n]\right) \quad \text{(J-16)}$$

Thus, the reduction in distortion incurred during a single coding pass may be computed by summing the outputs of one of two different functions, $f_s(.)$ or $f_m(.)$ as appropriate, whenever a sample becomes significant or its magnitude is refined and then scaling the result at the end of the coding pass by a constant value which is easily computed from the bit-plane index and the value of $\omega_i \Delta_i^2$. The argument to these functions, $v_i^p[m,n]$, has a binary representation of the form v.xxxxx, where v, the only bit before the binary point, is simply the value of magnitude bit p, i.e., $v_i^p[m,n]$. Exactly 6 extra bits beyond the binary point are used to index a 7-bit look-up table for $f_m(.)$ and a 6-bit look-up table for $f_s(.)$ (recall that we must have $1 \le v_i^p[m,n] < 2$ when a sample first becomes significant). Each entry of these look-up tables holds a 16-bit fixed point representation of $2^{13} f_s\left(v_i^p[m,n]\right)$ or $2^{13} f_m\left(v_i^p[m,n]\right)$, as appropriate, which means that the total distortion reduction associated with any given coding pass may be computed by accumulating these integer values into a 32-bit accumulator, without any risk of overflow.

J.14.4.2 Considerations for reversible transformations

Generally, the process for estimating distortion whilst encoding the coefficients produced by a reversible transformation is no different to that for a non-reversible transformation. There are, however, two subtle differences which must be pointed out here. Equations (J-15) and (J-16) are based upon the assumption that the dequantizer will represent each coefficient with the mid-point of the relevant quantization interval. This is the most likely behaviour for the quantizer most of the time, except for the least significant bit-plane in the reversible mode. In this case $\Delta_i = 1$, with no quantization error; midpoint reconstruction makes no sense here and the dequantizer represents the transform coefficients using the lower (in magnitude) threshold of the relevant quantization interval. Accordingly, Equations (J-15) and (J-16) should be modified to:

$$2^{2p}\omega_i \Delta_i^2 v_i^p[m,n]^2 = 2^{2p}\omega_i \Delta_i^2 f'_m\left(v_i^p[m,n]\right) \quad \text{(J-17)}$$

and:

$$2^{2p}\omega_i \Delta_i^2 \left(v_i^p[m,n] - 1\right)^2 = 2^{2p}\omega_i \Delta_i^2 f'_m\left(v_i^p[m,n]\right) \quad \text{(J-18)}$$

respectively.

J.15 Guidelines on handling YCC codestream

There are numerous applications and devices in both still and motion consumer imaging that cannot be considered without support for YCC and direct production of sub-sampled chrominance data. In such cases, the signalling of multiple component transformation within the codestream may not be necessary. This clause provides guidelines on how to handle YCC data.

However, it is not intended to imply that YCC data shall be sub-sampled.

J.15.1 Use of multiple component transformation

It is not necessary to use the multiple component transformation in order to support YCC data as the components are already decorrelated. Therefore, the multiple component transformation signal of the SGcod parameter defined in Table A.17 shall always be "0000 0000".

J.15.2 Using the JP2 format

There are devices that will automatically output component transformed YCC data in sYCC colourspace. The JP2 format supports these cases by specifying the EnumCS value to "18" as defined in Table I.10.

J.15.3 Chrominance offset

Chrominance offset (a common term for the sub-sampling of chrominance components and their relative offsets) is specified in a JPEG 2000 codestream using the CRG marker (see A.9.1). Figures J.13, J.14, J.15, and J.16 shows examples of well-known chrominance offset patterns. Table J.26 shows example SIZ (see A.5.1) and CRG maker segment parameter values for each pattern.

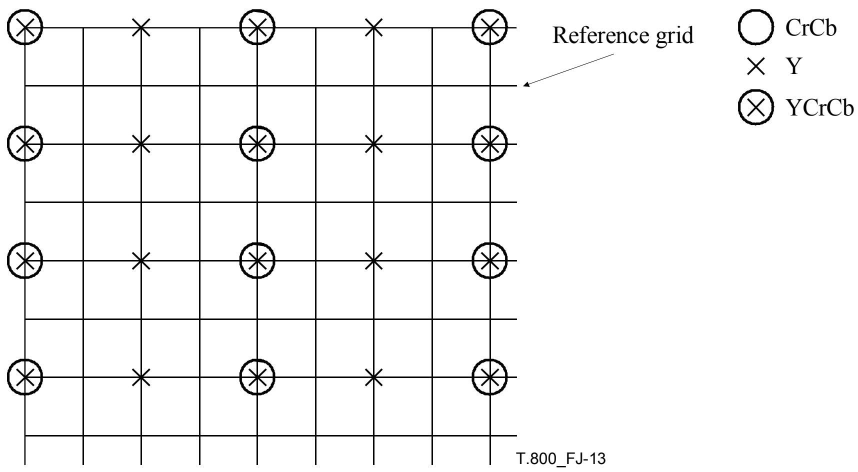


Figure J.13 – 4:2:2 format (co-sited)

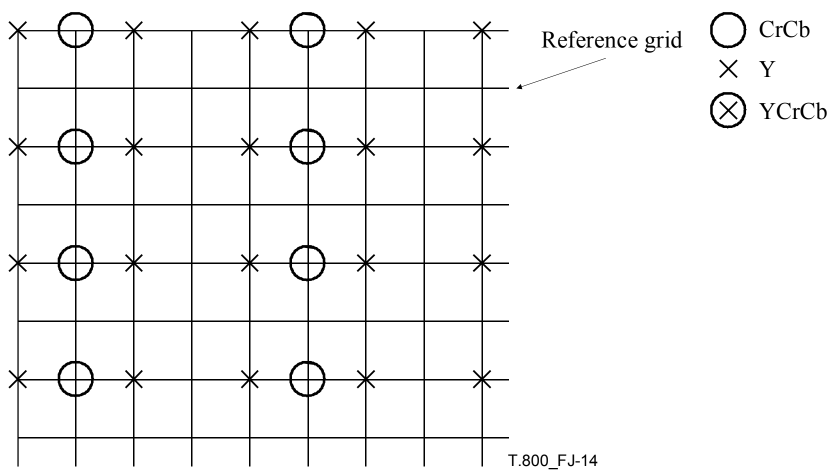


Figure J.14 – 4:2:2 format (centered)

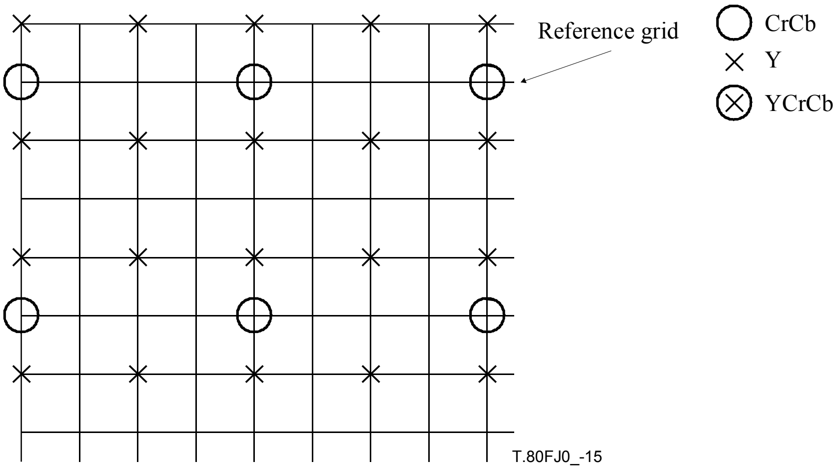


Figure J.15 – 4:2:0 format (co-sited)

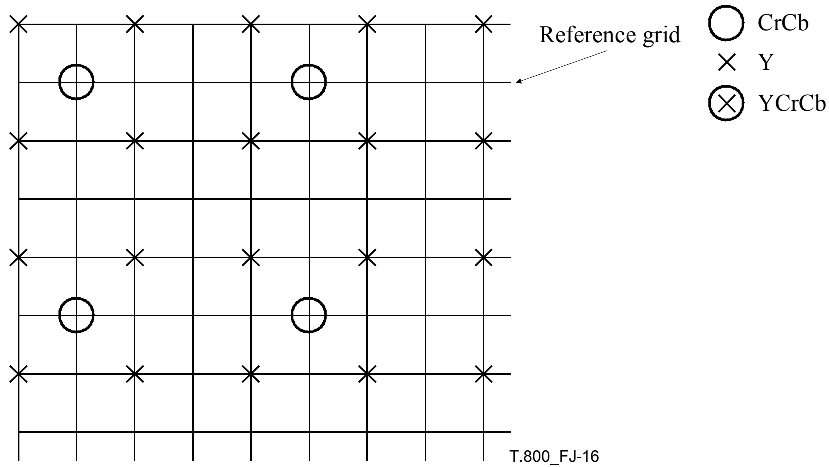


Figure J.16 – 4:2:0 format (centered)

Table J.26 – CRG (Component registration) values

		Figure J.13	Figure J.14	Figure J.15	Figure J.16
Y	(XRsiz, YRsiz)	(2, 2)	(2, 2)	(2, 2)	(2, 2)
	(XOsiz, YOsiz)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
	(Xcrg, Ycrg)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
Cb	(XRsiz, YRsiz)	(4, 2)	(4, 2)	(4, 4)	4, 4)
	(XOsiz, YOsiz)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
	(Xcrg, Ycrg)	(0, 0)	(16 384, 0)	(0, 16 384)	(16 384, 16 384)
Cr	(XRsiz, YRsiz)	(4, 2)	(4, 2)	4, 4)	4, 4)
	(XOsiz, YOsiz)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
	(Xcrg, Ycrg)	(0, 0)	(16 384, 0)	(0, 16 384)	(16 384, 16 384)
NOTE – The CRG value is defined so that all component samples shall be located in the reference grid points. Therefore, (XRsiz, YRsiz) = (2, 2) for Y does not mean samples of Y are also sub-sampled.					

Annex K

Bibliography

(This annex does not form an integral part of this Recommendation | International Standard)

K.1 General

[1] BOLIEK (M.), GORMISH (M.J.), SCHWARTZ (E.L.), KEITH (A.): Decoding compression with reversible embedded wavelets (CREW) codestreams, *Electronic Imaging*, Vol. 7, No. 3, July 1998.

[2] EBRAHIMI (T.), CHRISTOPOULOS (C.): JPEG 2000 standard: Still image compression scheme of 21st century, Tampere, Finland, 5-8 September 2000.

[3] MARCELLIN (M.W.), GORMISH (M.), BILGIN (A.), BOLIEK (M.): An Overview of JPEG-2000, *Proc. of IEEE Data Compression Conference*, Snowbird, Utah, March 2000.

[4] SKODRAS (A.), CHRISTOPOULOS (C.), EBRAHIMI (T.): JPEG2000: The upcoming still image compression standard, *(Invited paper) Proceedings of the 11th Portuguese Conference on Pattern Recognition (RECPAD 2000)*, pp. 359-366, Porto, Portugal, 11-12 May 2000.

[5] ZANDI (A.), ALLEN (J.D.), SCWHARTZ (E.L.), BOLIEK (M.): CREW: Compression with reversible embedded wavelets, *Proceedings of Data Compression Conference*, Snowbird, Utah, pp. 212-221, March 1995.

K.2 Quantization and entropy coding

[6] ORDENTLICH (E.), WEINBERGER (M.J.), SEROUSSI (G.): A low-complexity modelling approach for embedded coding of wavelet coefficients, *Proceedings of Data Compression Conference*, pp. 408-417, Snowbird, Utah, March 29-April 1, 1998.

[7] ORDENTLICH (E.), TAUBMAN (D.), WEINBERGER (M.J.), SEROUSSI (G.), MARCELLIN (M.): Memory Efficient Scalable Line-based Image Coding, *Proceedings of Data Compression Conference*, pp. 218-227, Snowbird, Utah, March 29-31, 1999.

[8] PENNEBAKER (W.), MITCHELL (J.): JPEG Still Image Data Compression Standard, *Van Nostrand Reinhold*, New York, 1993.

[9] SEMENTILLI (P.J.), BILGIN (A.), KASNER (J.H.), MARCELLIN (M.W.): Wavelet TCQ: Submission to JPEG 2000, (invited paper), *Proceedings, Applications of Digital Image Processing, SPIE*, San Diego, California, July 1998.

[10] TAUBMAN (D.), ZAKHOR (A.): Multirate 3-D subband coding of video, *IEEE Trans. Image Processing*, Vol. 3, pp. 572-88, September 1994.

[11] TAUBMAN (D.): High performance scalable image compression with EBCOT, *IEEE Trans. on Image Processing,* July 2000.

[12] WOODS (J.W.), NAVEEN (J.): A filter based bit allocation scheme for subband compression of HDTV, *IEEE Trans. on Image Processing*, July 1992.

K.3 Wavelet transformation

[13] ADAMS (M.D.), KOSSENTINI (F.): Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis, *IEEE Trans. on Image Processing*, 2000.

[14] ANTONINI (M.), BARLAUD (M.), MATHIEU (P.), DAUBECHIES (I.): Image coding using wavelet transform, *IEEE Trans. on Image Processing* Vol. 1, pp. 205-220, April 1992.

[15] BRISLAWN (C.M.): Classification of nonexpansive symmetric extension transforms for multirate filter banks, *Applied and Computational Harmonic Analysis*, Vol. 3, pp. 337-57, 1996.

[16] CALDERBANK (A.R.), DAUBECHIES (I.), SWELDENS (W.), YEO (B.-L.): Wavelet transforms that map integers to integers, *Applied and Computational Harmonic Analysis*, Vol. 5, No. 3, pp. 332-369, July 1998.

[17] CHRYSAFIS (C.), ORTEGA (A.), Line-Based, Reduced Memory, Wavelet Image Compression, *IEEE Trans. on Image Processing*, Vol. 9, No 3, pp. 378-389, March 2000.

[18] CHUI (C.K.): An Introduction to Wavelets, *Academic Press*, Boston, 1992.

[19] CHUI (C.K.): Wavelets: A Mathematical Tool for Signal Analysis, *SIAM Publ.*, Philadelphia, 1997.

[20] DAUBECHIES (I.): Ten Lectures on Wavelets, *SIAM Publ.*, Philadelphia, 1992.

[21] DAUBECHIES (I.), SWELDENS (W.): Factoring wavelet transforms into lifting steps, *Journal of Fourier Analysis and Applications*, Vol. 4, No. 3, pp. 247-269, 1999.

[22] LEGALL (D.), TABATABAI (A.): Subband 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.

[23] MALLAT (S.): A Wavelet Tour of Signal Processing, *Academic Press,* 2nd Edition, San Diego, 1999.

[24] SHENG (F.), BILGIN (A.), SEMENTILLI (P.J.), MARCELLIN (M.W.): Lossy and lossless image compression using reversible integer wavelet transforms, *Proceedings, IEEE International Conference on Image Processing*, Chicago, Illinois, October 1998.

[25] SWELDENS (W.): The lifting scheme: a custom-design construction of biorthogonal wavelets, *Applied and Computational Harmonic Analysis*, Vol. 3, No. 2, pp 186-200, 1996.

[26] SWELDENS (W.): The lifting scheme: A construction of second generation wavelets, *SIAM Journal on Mathematical Analysis*, Vol. 29, No. 2, pp 511-546, 1997.

K.4 Region of interest coding

[27] CHRISTOPOULOS (C.), ASKELOF (J.), LARSSON (M.): Efficient methods for encoding regions of interest in the upcoming JPEG2000 still image coding standard, *IEEE Signal Processing letters*, September 2000.

[28] ATSUMI (E.), FARVARDIN (N.): 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.

[29] NISTER (D.), 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,

[30] NISTER (D.), CHRISTOPOULOS (C.): Lossless region of interest with embedded wavelet image coding, *Signal Processing*, Vol. 78, No. 1, pp. 1-17, October 1999.

[31] SANTA CRUZ (D.), LARSSON (M.), ASKELOF (J.), EBRAHIMI (T.), CHRISTOPOULOS (C.): Region of Interest coding in JPEG2000 for interactive client/server applications, *IEEE International Workshop on Multimedia Signal Processing*, Copenhagen, Denmark, 13-15 September 1999.

K.5 Visual frequency weighting

[32] ALBANESI (M.), BERTOLUZZA (S.): 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.

[33] ECKERT (M.): Lossy compression using wavelets, block DCT, and lapped orthogonal transforms optimised with a perceptual model, *SPIE*, Vol. 3031, pp. 339-350, 1997.

[34] O'ROURKE (T.), STEVENSON (R.): Human visual system based wavelet decomposition for image compression, *J. VCIP V*. 6, pp. 109-121, 1995.

[35] WATSON (A.B.), YANG (G.), SOLOMON (J.), VILLASENOR (J.): Visibility of wavelet quantization noise, *IEEE Trans. on Image Processing*, Vol. 6, pp. 1164-1175, 1997.

K.6 Error resilience

[36] MAN (H.), KOSSENTINI (F.), SMITH (M.): 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.

[37] LIANG (J.), TALLURI (R.): Tools for Robust Image and Video Coding in JPEG2000 and MPEG-4 Standards, *Proceedings of SPIE Visual Communications and Image Processing Conference (VCIP)*, January 1999, San Jose, CA.

[38] MOCCAGATA (I.), SODAGAR (S.), LIANG (J.), CHEN (H.): Error Resilient Coding in JPEG-2000 and MPEG-4, *IEEE Journal of Selected Areas in Communications (JSAC)*, June 2000.

K.7 Scan-based coding

[39] LAMBERT-NEBOUT (C.), LATRY (C.), MOURY (G.), ANTONINI (M.), BARLAUD (M.), PARISOT (C.): On-Board Optical image compression for future high resolution space remote sensing systems, *Proc. SPIE*, Vol. 4115, 31 July-3 August 2000.

[40] FLOHR (T.J.), MARCELLIN (M.W.), ROUNTREE (J.C.): Scan-Based Processing with JPEG 2000, *Proc. SPIE*, Vol. 4115, 31 July-3 August 2000.

K.8 Colour

[41] International Color Consortium (ICC), *ICC Profile Format Specification 1:1998-09*, (1998).

[42] IEC 61966-2-1 (1999), *Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management – Default RGB colour space – sRGB*.

[43] LAM (K.M.): Metamerism and Colour Constancy, *University of Bradford,* 1985.

[44] LUO (M.R.), LO (M.C.), KUO (W.G.): The LLab(l:c) Colour Model, *Color Research and Application,* Vol. 21, pp. 412-429 (1996).

[45] LUO (M.R.), HUNT (R.W.G.): A Chromatic Adaptation Transform and a Colour Constancy Index, *Color Research and Appication,* Vol. 23, pp. 154-158, 1997.

[46] LUO (M.R.), HUNT (R.W.G.): The Structure of the CIE 1997 Colour Appearance Model (CIECAM97s), *Color Research and Application,* Vol. 23, 1997.

[47] NIELSEN (M.), STOKES (M.): The Creation of the sRGB ICC Profile, *Proceedings of the IS&T/SID 6th Color Imaging Conference,* pp. 253-257, 1998.

[48] GIORGIANNI (E.), MADDEN (T.): Digital Color Management: Encoding Solution, *Addision-Wesley*, 1998.

[49] SPAULDING (K.E.), WOOLFE (G.J.), GIORGIANNI (E.J.): Reference Input/Output Medium Metric RGB color encodings (RIMM/ ROMM RGB), *Proc. PICS 2000 Conference*, March 26-29, 2000, Portland, OR.

Annex L

Patent statement

(This annex does not form an integral part of this Recommendation | International Standard)

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this Recommendation | International Standard may involve the use of patents, as indicated in Table L.1.

Table L.1 – Received intellectual property rights statements

Number	Company
1	Algo Vision plc
2	Canon Incorporated
3	Digital Accelerator Corporation
4	Telefonaktiebolaget LM Ericsson
5	Hewlett Packard Company
6	International Business Machines, Inc.
7	LizardTech, Incorporated
8	LuraTech
9	Mitsubishi Electric Corporation
10	PrimaComp Incorporated
11	Ricoh Company, Limited
12	Sarnoff Corporation
13	Sharp Corporation
14	TeraLogic Incorporated
15	University of Arizona
16	Washington State University
17	Sony Corporation

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

The holders of these patent rights have assured the ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patents right are registered with ISO and IEC.

Attention is drawn to the possibility that some of the elements of this Recommendation | International Standard may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Index

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	**Terminals for telematic services**
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems

Geneva, 2002