ISO/IEC JTC 1/SC 29/WG 1 N1646R

Date: 16 March 2000

ISO/IEC JTC 1/SC 29/WG 1
(ITU-T SG8)

Coding of Still Pictures

JBIG
Joint Bi-level Image Experts Group

JPEG
Joint Photographic Experts Group

TITLE: JPEG 2000 Part I Final Committee Draft Version 1.0

SOURCE: ISO/IEC JTC1/SC29 WG1, JPEG 2000 Editor Martin Boliek, Co-editors Charilaos Christopoulos, and Eric Majani

PROJECT: 1.29.15444 (JPEG 2000)

STATUS: New pdf file distilled 11 April 2000

REQUESTED ACTION: For review and action as appropriate

DISTRIBUTION: WG1

Contact:

Convenor, ISO/IEC JTC 1/SC 29/WG 1
Dr. Daniel T. Lee
Hewlett-Packard Company,
11000 Wolfe Road, MS 42UO
Cupertino, CA 95014, USA
Tel: 1 408 447 4160, Fax: 1 408 447 2842 fax
Internet: daniel_lee@hp.com

Editor, 15444 (JPEG-2000)
Mr. Martin Boliek
Ricoh Silicon Valley, Inc.
2882 Sand Hill Road, Suite 115
Menlo Park, CA 94025, USA
Tel: 1 650 496 5705, Fax: 1 650 854 8740
Internet: boliek@rsv.ricoh.com

INFORMATION TECHNOLOGY

JPEG 2000 IMAGE CODING SYSTEM

JPEG 2000 FINAL COMMITTEE DRAFT VERSION 1.0, 16 MARCH 2000

THE ISO AND ITU WILL PROVIDE COVER PAGES.

CONTENTS

Foreword 11
Introduction 11
1 Scope 1
2 References 1
2.1 Identical Recommendations | International Standards 1
2.2 Additional references 1
3 Definitions 2
4 Abbreviations 6
5 Symbols 7
6 General description 8
6.1 Purpose 8
6.2 Codestream 8
6.3 Coding principles 9
7 Encoder requirements 11
8 Decoder requirements 11
8.1 Codestream syntax requirements 11
8.2 Optional file format requirements 11
Annex A Codestream syntax 13
A.1 Headers and marker segments 13
A.2 Information in the marker segments 15
A.3 Construction of the codestream 17
A.4 Delimiting markers 21
A.5 Fixed information marker segment 26
A.6 Functional marker segments 29
A.7 Pointer marker segments 42
A.8 In bit stream marker segments 49
A.9 Informational markers 51
Annex B Data ordering 53
B.1 Image division into components 53
B.2 Image division into tiles and tile-components 54
B.3 Example of the mapping of components to the reference grid 55
B.4 Tile-component division into resolutions and sub-bands 58
B.5 Division of resolutions into precincts 59
B.6 Division of the sub-bands into code-blocks 60
B.7 Layers 61
B.8 Packets 61
B.9 Packet header information coding 61
B.10 Tile Data and Tile-Parts 67

B.11 Progression Order. 67
Annex C Arithmetic entropy coding. 71
C.1 Binary encoding (informative). 71
C.2 Description of the arithmetic encoder (informative) 72
C.3 Arithmetic decoding procedure 84
Annex D Coefficient bit modeling 93
D.1 Code-block scan pattern within code-blocks 93
D.2 Coefficient bits and significance 93
D.3 Decoding passes over the bit-planes 93
D.4 Initializing and terminating 98
D.5 Error resilience segmentation symbol 100
D.6 Selective arithmetic decoding bypass 100
D.7 Vertically causal context formation 102
D.8 Flow diagram of the code-block coding. 102
Annex E Quantization 105
E.1 Scalar coefficient dequantization (normative) 105
E.2 Scalar coefficient quantization (informative). 107
Annex F Discrete wavelet transformation of tile components 109
F.1 Introduction and overview 109
F.2 The inverse discrete wavelet transformation (normative) 109
F.3 Forward Transformation (informative) 119
Annex G DC level shifting and component transformations 129
G.1 DC level shifting of tile components 129
G.2 Reversible component transformation (RCT) 130
G.3 Irreversible component transformation (ICT). 130
G.4 Chrominance component sub-sampling and the image reference grid (informative) 131
Annex H Coding of images with Regions of Interest 133
H.1 Description of the Maxshift method 133
H.2 Region of interest mask generation 134
H.3 Remarks on Region of Interest coding. 135
H.4 An example of the interpretation of the Maxshift method (Informative) 136
Annex I JP2 file format syntax 137
I.1 File format scope 137
I.2 File format definitions 137
I.3 File format normative references 138
I.4 Introduction 138
I.5 Greyscale/Colour/Palettized/multi-component specification architecture. 140
I.6 Box definition. 142
I.7 Defined boxes. 144

I.8 Adding intellectual property rights information in JP2 157
I.9 Adding vendor specific information to the JP2 file format 157
I.10 Dealing with unknown boxes 160
Annex J Examples and Guidelines 161
J.1 Software Conventions Adaptive Entropy Decoder 161
J.2 Row-based wavelet transform 163
J.3 Scan-Based Coding 170
J.4 Error resilience 170
J.5 Implementing the Restricted ICC method outside of a full ICC colour management engine 171
J.6 An example of the interpretation of multiple components 174
J.7 An example of decoding showing intermediate steps 174
J.8 Visual Frequency Weighting 177
J.9 Encoder sub-sampling of components 179
J.10 Rate control 180
Annex K Bibliography 185
K.1 General 185
K.2 Wavelet transform 185
K.3 Quantization and Entropy coding 185
K.4 Region of Interest coding 185
K.5 Visual frequency weighting 186
K.6 Error resilience 186
K.7 Scan-based coding 186
K.8 ICC colour 186
Index 187
Annex L Patent Statement 189

LIST OF FIGURES

Figure 6-1 Specification block diagram 9
Figure A-1 Example of the marker segment description figures 15
Figure A-2 Construction of the codestream 18
Figure A-3 Construction of the main header 19
Figure A-4 Construction of a tile-part header 20
Figure A-5 Start of tile-part syntax 22
Figure A-6 Image and tile size syntax 26
Figure A-7 Coding style default syntax 29
Figure A-8 Coding style component syntax. 33
Figure A-9 Coding style default syntax 35
Figure A-10 Quantization default syntax 37
Figure A-11 Quantization component syntax 39
Figure A-12 Progression order change, tile syntax 40
Figure A-13 Coded tile-part lengths 42
Figure A-14 Tile-part length, main header syntax 42
Figure A-15 Packets length, main header syntax 44
Figure A-16 Packet length, tile-part header syntax 45
Figure A-17 Packed packet headers, main header syntax 46
Figure A-18 Packed packet headers, tile-part header syntax 48
Figure A-19 Start of packet syntax 49
Figure A-20 Coding style component syntax. 51
Figure B-1 Reference grid diagram 53
Figure B-2 Tiling of the reference grid diagram 54
Figure B-3 Reference grid example 56
Figure B-4 Example tile sizes and locations for component 0 57
Figure B-5 Example tile sizes and locations for component 1 58
Figure B-6 Precinct partition 59
Figure B-7 Code-blocks in sub-band b 60
Figure B-8 Example of a tag tree representation 62
Figure B-9 Example of the information known to the encoder 65
Figure B-10 Example of progression order change in two dimensions 70
Figure C-1 Arithmetic encoder inputs and outputs 71
Figure C-2 Encoder for the MQ-coder. 73
Figure C-3 ENCODE procedure 74
Figure C-4 CODE1 procedure 74
Figure C-5 CODE0 procedure 75
Figure C-6 CODELPS procedure with conditional MPS/LPS exchange 76
Figure C-7 CODEMPS procedure with conditional MPS/LPS exchange 79
Figure C-8 Encoder renormalisation procedure. 80
Figure C-9 BYTEOUT procedure for encoder 81
Figure C-10 Initialisation of the encoder 82
Figure C-11 FLUSH procedure 83
Figure C-12 Setting the final bits in the C register 84
Figure C-13 Arithmetic decoder inputs and outputs 84

Figure C-14 Decoder for the MQ-coder 85
Figure C-15 Decoding an MPS or an LPS 86
Figure C-16 Decoder MPS path conditional exchange procedure 87
Figure C-17 Decoder LPS path conditional exchange procedure 88
Figure C-18 Decoder renormalisation procedure 89
Figure C-19 BYTEIN procedure for decoder 90
Figure C-20 Initialisation of the decoder 91
Figure D-1 Example code-block scan pattern of a code-block 93
Figure D-2 Neighbors states used to form the context 94
Figure D-3 Flow chart for all coding passes on a code-block bit-plane 104
Figure E-1 Analysis gain of each sub-band of the wavelet transform decomposition 106
Figure E-2 Nominal dynamic rangefor each sub-band of the wavelet transform decomposition, where is the bit depth of the original image tile-component 106
Figure F-1 Inputs and outputs of the IDWT procedure 110
Figure F-2 The IDWT (NL=2) 110
Figure F-3 The IDWT Procedure 111
Figure F-4 Inputs and outputs of the 2D_SR procedure 111
Figure F-5 One-level recomposition from four sub-bands (2D_SR procedure) 111
Figure F-6 The 2D_SR procedure 112
Figure F-7 Parameters of 2D_INTERLEAVE procedure 112
Figure F-9 Inputs and outputs of the HOR_SR procedure 112
Figure F-8 The 2D_INTERLEAVE procedure 113
Figure F-10 The HOR_SR procedure 114
Figure F-11 Inputs and outputs of the VER_SR procedure 114
Figure F-12 The VER_SR procedure 115
Figure F-13 Parameters of the 1D_SR procedure 115
Figure F-14 The 1D_SR procedure 116
Figure F-15 1D_EXTR procedure implementing periodic symmetric extension 117
Figure F-16 Periodic symmetric extension of signal 117
Figure F-17 Parameters of the 1D_IFILTR procedure 118
Figure F-18 Inputs and outputs of the FDWT procedure 119
Figure F-19 The FDWT (NL=2) 120
Figure F-20 The FDWT Procedure 120
Figure F-21 Inputs and outputs of the 2D_SD procedure 121
Figure F-22 One-level decomposition into four sub-bands (2D_SD procedure) 121
Figure F-23 The 2D_SD procedure 121
Figure F-24 Inputs and outputs of the VER_SD procedure 122
Figure F-25 The VER_SD procedure 122
Figure F-26 Inputs and outputs of the HOR_SD procedure 123
Figure F-27 The HOR_SD procedure 123
Figure F-28 Parameters of 2D_DEINTERLEAVE procedure 123
Figure F-29 The 2D_DEINTERLEAVE procedure 124
Figure F-30 Parameters of the 1D_SD procedure 125
Figure F-31 The 1D_SD procedure 125
Figure F-32 Parameters of the 1D_FILTD procedure 125
Figure G-1 Placement of the DC level shifting with component transform 129

Figure G-2 Placement of the DC level shifting without component transform 129
Figure H-1 The inverse 5x3 transform 135
Figure H-2 The inverse 9x7 transform 135
Figure I-1 Conceptual structure of a JP2 file 139
Figure I-2 Organization of an Box 142
Figure I-3 Illustration of box lengths 143
Figure I-4 Organization of the contents of a Profile box 145
Figure I-5 Organization of the contents of a JP2 header box 146
Figure I-6 Organization of the contents of an Image Header box 146
Figure I-7 Organization of the contents of a BitsPerComponent box 148
Figure I-8 Organization of the contents of a Colour Specification box 148
Figure I-9 Organization of the contents of the Palette box 150
Figure I-10 Organization of the contents of a Component Definition box 152
Figure I-11 Organization of the contents of the Resolution box 154
Figure I-12 Organization of the contents of the Capture Resolution box 155
Figure I-13 Organization of the contents of the Default Display Resolution box 156
Figure I-14 Organization of the contents of the Contiguous codestream box 157
Figure I-15 Organization of the contents of the XML box 157
Figure I-16 Organization of the contents of the UUID box 158
Figure I-17 Organization of the contents of a UUID Info box 158
Figure I-18 Organization of the contents of a UUID Info box 159
Figure I-19 Organization of the contents of a URL box 159
Figure J-1 Initialisation of the software-conventions decoder 161
Figure J-2 Decoding an MPS or an LPS in the software-conventions decoder 162
Figure J-3 Inserting a new byte into the C register in the software-conventions decoder 163
Figure J-4 The FDWT_ROW procedure 164
Figure J-5 The GET_ROW procedure 165
Figure J-6 The INIT procedure 165
Figure J-7 The START_VERT procedure 166
Figure J-8 The RB_VERT_1 procedure 167
Figure J-9 The RB_VERT_2 procedure 167
Figure J-10 The END_1 procedure 168
Figure J-11 The END_2 procedure 169
Figure J-12 Illustration of code-block contributions to bit-stream layers 181

LIST OF TABLES

Table A-1 Marker definitions 14
Table A-2 List of marker segments 15
Table A-3 Information in the marker segments 16
Table A-4 Start of codestream parameter values 21
Table A-5 Start of tile-part parameter values 22
Table A-6 Number of tile-parts, TNsot, parameter value 22
Table A-7 Start of data parameter values 24
Table A-8 End of codestream parameter values 25
Table A-9 Image and tile size parameter values 27
Table A-10 Capability Rsiz parameter 27
Table A-11 Component Ssiz parameter 27
Table A-12 Coding style default parameter values 29
Table A-13 Coding style parameter values for the Scod parameters 30
Table A-14 Coding style parameter values of the SPcod parameters 30
Table A-15 Progression order for the SPcod, Ppod parameters 30
Table A-16 Width and height of the code-blocks for the SPcod and SPcoc parameters 31
Table A-17 Code-block style for the SPcod and SPcoc parameters 31
Table A-18 Transform for the SPcod and SPcoc parameters 31
Table A-19 Multiple component transformation CSsiz parameter 32
Table A-20 Packet partition width and height for the SPcod parameters 32
Table A-21 Coding style component parameter values 33
Table A-22 Coding style parameter values for the Scoc parameters 34
Table A-23 Coding style parameter values from SPcoc parameters 34
Table A-24 Region-of-interest parameter values 35
Table A-25 Region-of-interest parameter values for the Srgn parameter 35
Table A-26 Region-of-interest values from SPrgn parameter (Srgn = 0) 36
Table A-27 Quantization default parameter values 37
Table A-28 Quantization default values for the Sqcd and Sqcc parameters 37
Table A-29 Reversible step size values for the SPqcd and SPqcc parameters (5-3 transform only) 38
Table A-30 Quantization values for scalar quantization for the SPqcd and SPqcc parameters (9-7 transform only) . . 38
Table A-31 Quantization component parameter values 39
Table A-32 Progression order change, tile parameter values 40
Table A-33 Tile-part length, main header parameter values 43
Table A-34 Size parameters for Stlm 43
Table A-35 Packets length, main header parameter values 44
Table A-36 Iplm, Iplt list of packet lengths 44
Table A-37 Packet length, tile-part headers parameter values 45
Table A-38 Packed packet headers, main header parameter values 46
Table A-39 Index for the PPM marker segment parameters for Zppm 47
Table A-40 Packet header, tile-part headers parameter values 48
Table A-41 Start of packet parameter values 49
Table A-42 End of packet header parameter values 50
Table A-43 Comment and extension parameter values 51
Table A-44 Registration values for the Rcme parameter 51

Table B-1 Quantities (xo_b,yo_b) for sub-band b. 59
Table B-2 Codewords for the number of coding passes for each code-block 63
Table B-3 Example packet header bit stream 65
Table C-1 Encoder register structures 73
Table C-2 Qe values and probability estimation process 76
Table C-3 Decoder register structures 85
Table D-1 Contexts for the significance propagation pass and cleanup coding passes 94
Table D-2 Contributions of the vertical (and the horizontal) neighbors to the sign context 95
Table D-3 Sign contexts from the vertical and horizontal contributions 96
Table D-4 Contexts for the magnitude refinement coding passes 97
Table D-5 Run-length decoder for cleanup passes 97
Table D-6 Example of sub-bit-plane coding order and significance propagation 98
Table D-7 Initial states for all contexts. 98
Table D-8 Examples of arithmetic coder termination patterns. 99
Table D-9 Selective arithmetic coding bypass 101
Table D-10 Decisions in the context model flow chart 102
Table D-11 Coding in the context model flow chart 103
Table E-1 Impulse response of the low and high pass synthesis filter for the 9-7 wavelet transform. 108
Table F-1 Extension to the left. 117
Table F-2 Extension to the right. 118
Table I-1 Binary structure of an box 142
Table I-2 Boxes defined within this Recommendation | International Standard 143
Table I-3 Format of the contents of the Profile box 145
Table I-4 Format of the contents of the Image Header box 147
Table I-5 Format of the contents of the BitsPerComponent box 148
Table I-6 Legal METH values 149
Table I-7 Legal EnumCS values 150
Table I-8 Format of the contents of the Colr box 150
Table I-9 Format of the contents of the Palette box 151
Table I-10 Typ^i field values 152
Table I-11 Asoci field values 153
Table I-12 Colours indicated by the $Asoc^i$ field 153
Table I-13 Component definition & ordering data structure values 154
Table I-14 Format of the contents of the Capture resolution box 155
Table I-15 Format of the contents of the Default display resolution box 156
Table I-16 Format of the contents of the Contiguous codestream box 157
Table I-17 Format of the contents of a UUID box 158
Table I-18 UUID List box contents data structure values. 159
Table I-19 URL box contents data structure values 159
Table J-1 Error resilience tools 170
Table J-2 Recommended frequency weighting 179
Table L-1 Received intellectual property rights statements. 189

Foreword

Forward to be supplied by ISO and ITU.

Introduction

Introduction to be supplied by ISO and ITU.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – JPEG 2000 IMAGE CODING SYSTEM

1 Scope

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

This Recommendation | International Standard

- specifies decoding processes for converting compressed image data to reconstructed image data
- specifies a codestream syntax containing information for interpreting the compressed image data
- specifies a file format
- provides guidance on encoding processes for converting source image data to compressed image data
- provides guidance on how to implement these processes in practice

2 References

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

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation T.81 | ISO/IEC 10918-1:1994, Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines.
- ITU-T Recommendation T.88 | ISO/IEC 14492-1, Lossy/lossless coding of bi-level images

2.2 Additional references

- Coded character set—7 bit, American Standard Code for Information Interchange, ANSI X3.4–1986.
- ISO/IEC 646:1991, ISO 7-bit coded character set for information interchange.
- ITU-T Recommendation T.83 | ISO/IEC 10918-2:1995, Information technology - Digital compression and coding of continuous-tone still images: Compliance testing.
- ITU-T Recommendation T.84 | ISO/IEC 10918-3:1996, Information technology - Digital compression and coding of continuous-tone still images: Extensions.
- ITU-T Recommendation T.84 | ISO/IEC 10918-3 Amd 1 (In preparation), Information technology - Digital compression and coding of continuous-tone still images: Extensions - Amendment 1.

- ITU-T Recommendation T.86 | ISO/IEC 10918-4, Information technology - Digital compression and coding of continuous-tone still images: Registration of JPEG Profiles, SPIFF Profiles, SPIFF Tags, SPIFF colour Spaces, APPn Markers, SPIFF, Compression types and Registration authorities (REGAUT).
- ITU-T Recommendation T.87 | ISO/IEC 14495-1, Lossless and near-lossless compression of continuous-tone still images-baseline.
- ITU-T Recommendation T.82 | ISO/IEC 11544:1994, Information technology - Coded representation of picture and audio information — Progressive bi-level image compression
- ISO 5807:1985, Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.
- International Color Consortium, ICC profile format specification. ICC.1:1998–09
- International Electrotechnical Commission. Color management in multimedia systems: Part 2: Colour Management, Part 2–1: Default RGB colour space—sRGB. IEC 61966–2–1 1998. 9 October 1998.
- W3C, Extensible Markup Language (XML 1.0), Rec-xml-19980210

3 Definitions

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

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

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

arithmetic coder: An entropy coder that converts variable length strings to variable length codes.

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

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

box contents: Refers to the data wrapped within the box structure. The contents of a particular box are stored within the DBox field within the Box data structure as defined in Annex I.6

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 as defined in Annex I.6.

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.

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. It does include any packet headers and in stream markers and marker segments not found in the main or tile-part headers.

big endian: The bits occur in order from most significant to least significant.

byte: Eight-bit octet.

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

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

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

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

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

coding pass: A complete pass through a code-block where the appropriate coefficient values and context are applied. There are three types of coding passes: significance propagation pass, magnitude refinement pass and cleanup pass. The result of each pass (after arithmetic coding) is a stream of compressed data.

colour component: A component from the codestream that functions as an input to a colour transformation system. For example, a red component or a greyscale component would be a colour component.

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

component: A two-dimensional array of samples. A colour image typically consists of several components from a specified colour space, for instance representing red, green, and blue.

compressed data: Any data that is part of the bit stream except for packet headers and in stream markers and marker segments.

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

container box: An 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 container box. When used as part of a relationship between two boxes, the term container box refers to the box which directly contains the other box.

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

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

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

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

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

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

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

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

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

encoder: An embodiment of an encoding process.

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

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

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

functional markers and marker segments: Markers and marker segments that offer information about the decoding procedures to be used

header: Part of the codestream that contains only markers and marker segments. There are two types of headers. The main header is found at the beginning of the codestream and tile-part headers are found at the beginning of each tile-part.

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

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

image: The set of all components.

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

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

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

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

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

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

JP2 file: The name of file in the file format described in this specification. Structurally, a JP2 file is a contiguous sequence of boxes.

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

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

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

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

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

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

lossy: A descriptive term for encoding and decoding processes that are not lossless. Distortion free restoration is not assured. This includes both systems that are irreversible and those that include quantization.

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

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

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

marker segment: A marker and associated set of parameters.

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

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

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

packet partition: A division of one tile-component by a rectangular grid. One packet partition size is specified for each resolution level.

packet partition location: One rectangular region of a packet partition.

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

precinct: A sub-division of a tile-component, within a each resolution, used for limiting the size of packets.

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

progressive: 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 pixel fidelity.

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

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 line. After the end of the line is reached the next data point in the sequence is the left-most data point immediately below the current line. This order is continued to the end of the array.

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

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

reference grid: A regular rectangular array of points to which images, components, tiles, sub-bands, etc. are associated. Reference grid units or points are used to describe the mapping of the tiles and the components.

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

region of interest (ROI): A defined area of the image, component, or tile-component that is considered of particular relevance by some user defined measure.

resolution: The spatial mapping of samples to a physical space. In this Recommendation | International Standard the decomposition levels of the wavelet transform relate to each other with relative resolutions differing by powers of two.

reversible: A transformation, progression, system, or quantization that does not suffer systemic or quantization error and, therefore, allows lossless signal recovery. The result of reversible process may be lossy or lossless depending on the quantization and other factors in the system.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

tile-component: All the samples of a given component in a tile. There is a tile-component for every component and every tile.

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

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

tile-part 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.

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

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

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

XOR: Exclusive OR logical operator.

4 Abbreviations

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

ASCII: American Standard Code for Information Interchange

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

ICC: International Colour Consortium

IEC: International Electrotechnical Commission

ISO: International Organization for Standardization

ITTF: Information Technology Task Force

ITU: International Telecommunication Union

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

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

JURA: JPEG Utilities Registration Authority

1D-DWT: One-dimensional discrete wavelet transform

FDWT: Forward discrete wavelet transform

IDWT: Inverse of the forward discrete wavelet transform

LSB: Least significant bit.

MSB: Most significant bit.

PCS: Profile Connection Space

ROI: Region-of-interest

SNR: Signal to noise ratio.

UCS: Universal Character Set

URI: Uniform Resource Identifier

URL: Uniform Resource Location

UTF-8: UCS Transformation Format 8

UUID: Universal Unique Identifier

W3C: World-Wide Web Consortium

5 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.

CME: Comment and extension marker

COC: Coding style component marker

COD: Coding style default maker

EPH: End of packet header marker

EOI: End of image marker

PLM: Packet length, main header marker

PLT: Packet length, tile-part header marker

POD: Progression order change, default 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: Size of image marker

SOC: Start of image (codestream) marker

SOP: Start of partition marker

SOS: Start of scan marker

SOT: Start of tile marker

TLM: Tile length marker

6 General description

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

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

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

6.1 Purpose

There are 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 compressed image data.

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

Codestream syntax: A compressed image data representation that includes all parameter specifications used in the encoding 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.

6.2 Codestream

In general, this standard deals with three domains: spatial (samples) transformed (coefficients) and coded 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 coded 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.

The codestream is a linear stream of bits from the first bit to the last bit. For convenience, it can be divided into (8 bits) 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. This should be clarified in the text.

6.3 Coding principles

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

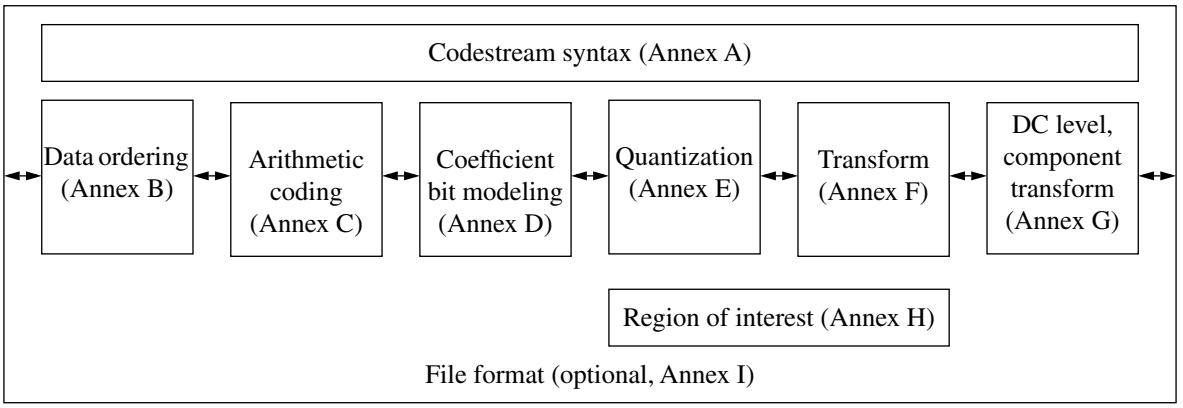


Figure 6-1 — Specification block diagram

Many images have multiple components. This specification has a facility to decorrelate three component planes. This is the only function in this specification that relates components to each other. (See Annex G.)

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

The tile-components are decomposed into different decomposition levels using a wavelet transform. These decomposition levels contain a number of sub-bands populated with coefficients that describe the horizontal and vertical spatial frequency characteristics of the original tile-component planes. The coefficients provide frequency information about a local area, rather than across the entire image like the Fourier Transform. That is, a small number of coefficients completely describes a single sample. A decomposition level is related to the next decomposition level by spatial powers of two. That is, each successive decomposition level of the sub-bands 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 Annex C, Annex D, and Annex B.)

The individual sub-bands of a tile-component are further divided into code-blocks. These rectangular arrays of coefficients can be extracted independently. The individual bit-planes of the coefficients in a code-block are coded with three coding passes. Each of these coding passes collects contextual information about the bit-plane data. (See Annex 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 data.

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

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

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

Several mechanisms are provided to allow the detection and concealment of bit errors that might occur over a noisy transmission channel. (See Annex D.5.)

The compressed data 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 also contain other data besides the codestream. (See Annex I.)

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

- The image is decomposed into components.
- The image and its components are decomposed into rectangular tiles. The tile-component is the basic unit of the original or reconstructed image.
- Performing the wavelet transform on a tile-component creates decomposition levels. These decomposition levels can create components with different resolutions.
- 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.
- The bit-planes of the coefficients in a code-block are entropy coded in three coding passes.
- Some of the coefficients can be coded first to provide a region of interest.

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

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

7 Encoder requirements

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

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

8 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Annex C through Annex 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.

8.1 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 parameters values of the syntax described in Annex A are not intended to portray the capabilities required to be compliant.

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.

8.2 Optional file format requirements

Annex I describes the optional file format contains meta-data about the image in addition to the codestream, which 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.

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.

Annex A

Codestream syntax

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

This Annex specifies the marker and marker segment syntax defined by this Recommendation | International Standard. These 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 Annex A are not intended to portray the capabilities required to be compliant.

A.1 Headers and marker segments

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

Headers are collections of markers and marker segments. There are two types of headers in this specification. 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.

A.1.1 Markers and marker segments

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 reserve unless specifically used.

A marker segment includes a marker and associated parameters, called marker parameters. In every marker segment the first two bytes after the marker shall be an unsigned big endian integer value that denotes the length in bytes of the marker parameters (including two bytes of this length parameter but not the two bytes of the marker itself).

A.1.2 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 marker and marker segments must be used to frame the headers and the data. Fixed information marker segments give required information about an image. The location of these marker segments, like delimiting marker segments, is specified. Functional marker segments are used to describe the coding functions used. In bit stream markers and marker segments are used for error resilience. Pointer marker segments point to specific offsets in the bit stream. Informational marker segments provide ancillary information.

A.1.3 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. Some of the markers are exactly the same. Those that are not have numbers that were reserved in ITU.T Rec. 81 | IS 10918-1, ITU-T Rec. T.84 | ISO/IEC 10918-3, and ITU-T Rec. T.87 | ISO/IEC 14495-1 are registered by the registration process defined in ITU-T Rec. T.86 | ISO/IEC 10918-4.

The marker range 0xFF30 — 0xFF3F is reserved by this specification for markers without marker parameters. This will enable backward compatibility. Table A-1 shows in which specification these markers and marker segments are defined.

Table A-1 — Marker definitions

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

A.1.4 Marker and marker segment and codestream rules

— Marker segments, and therefore the headers, are a multiple of 8 bits (one byte). Further, the bit stream data between the headers are padded to also be aligned to a multiple of 8 bits.

— All markers and marker segments in a tile-part header apply only to the tile to which it belongs.

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

— Delimiting and fixed information 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 accordingly.

— All parameter values in marker segments are big endian (most significant byte first).

— All markers with the marker value between 0xFF30 and 0xFF3F have no marker parameters.

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

A.1.5 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 a three letter abbreviation. The parameter values have capital letter designations with the marker's abbreviation as a subscript. A rectangle is used to indicate the parameters in the marker segment. The width of the rectangle is proportional to the number of bytes in the

field. A shaded rectangle (diagonal stripes) indicates that the parameter is of varying size. Two parameters with superscripts and a gray area between indicate a run of several of these parameters.

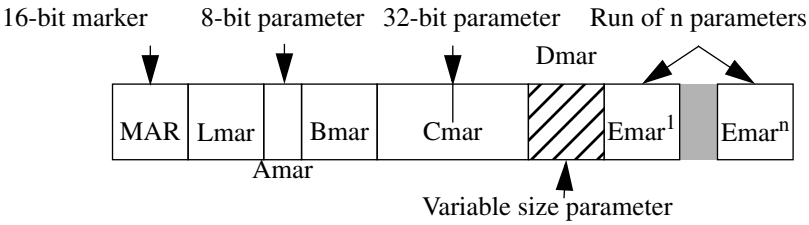


Figure A-1 — Example of the marker segment description figures

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 name MAR. The second rectangle represents the length parameter. Parameters Amar, Bmar, Cmar, and Dmar are 8, 16, 32 bit and variable length respectively. The parameter $Emar^i$ has a run from 1 to n.

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 theses parameters are described by a bit value in a bit field. In this case, the bits that do not matter for this parameter are denoted with an "x."

Some marker segments are described using the notation "Sxxx" and "SPxxx" (for a marker named 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 the information provided by the syntax and which marker segment contains that information.

Table A-2 — List of marker segments

	Name	Code	Main header[a]	Tile-part header[a]
Delimiting marker segments				
Start of codestream	SOC	0xFF4F	required	not allowed
Start of tile-part	SOT	0xFF90	not Allowed	required
Start of data	SOD	0xFF93	not allowed	last marker
End of codestream[b]	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

Table A-2 — List of marker segments

	Name	Code	Main header[a]	Tile-part header[a]
Region-of-interest	RGN	0xFF5E	optional	optional
Quantization default	QCD	0xFF5C	required	optional
Quantization component	QCC	0xFF5D	optional	optional
Progression order default	POD	0xFF5F	optional[c]	optional[c]
Pointer marker segments				
Tile-part lengths, main header	TLM	0xFF55	optional	not allowed
Packet length, main header	PLM	0xFF57	optional	not allowed
Packet length, tile-part header	PLT	0xFF58	not allowed	optional
Packed packet headers, main header	PPM	0xFF60	optional[d]	not allowed
Packed packet headers, tile-part header	PPT	0xFF61	not allowed	optional[d]
In bit stream marker segments				
Start of packet	SOP	0xFF91	not allowed	optional, in bit stream
End of packet header	EPH	0xFF92	not allowed	optional, in bit stream
Informational marker segments				
Comment and extension	CME	0xFF64	optional	optional

a. Required means the marker segment shall be in this header, optional means it may be used.
b. The EOC marker is the last in the codestream. It is in neither the main nor the tile-part headers.
c. The POD marker segment is required if there are progression order changes.
d. Either the PPM or PPT marker segment is required if the packet headers are not distribuited in the bit stream. If the PPM marker segment is used then PPT marker segments shall not be used, and visa versa.

Table A-3 — Information in the marker segments

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

Table A-3 — Information in the marker segments

Information	Marker segment
Coding style Number of decomposition levels Progression order Number of layers Code-block size Code-block style Wavelet transform	COD, COC
Region of interest shift	RGN
No quantization Quantization implicit Quantization explicit	QCD, QCC
Progression starting point Progression ending point Progression order default	POD
Error resilience End of packet header	SOP, EPH
Code-block values for new layers Code-block layer number Code-block inclusion Maximum bit depth Truncation point Bit stream length for decomposition level and layer in a code-block	packet header, PPM, PPT
Packet lengths	PLM, PLT
Optional information	CME

A.3 Construction of the codestream

Figure A-2 shows the construction of the codestream. Figure A-3 shows the main header construction. Note that all of the solid lines show required marker segments. The marker segments on the left are required to be in a specific location. The

dashed lines show optional or possibly not required marker segments. Figure A-4 shows the construction of a tile-part header.

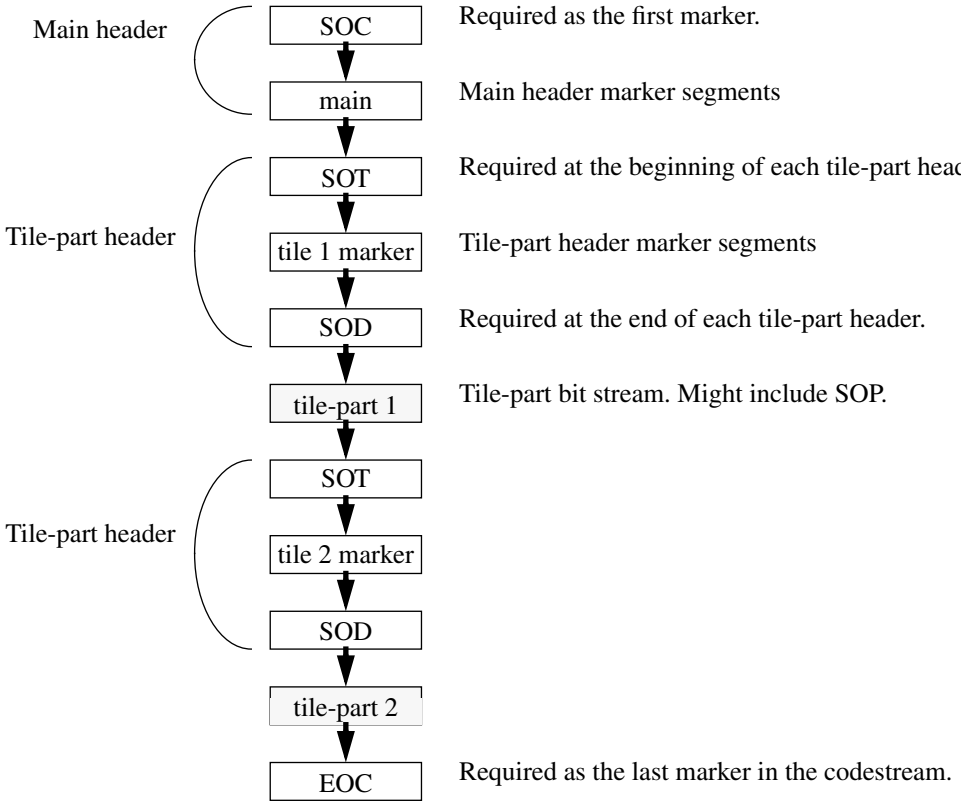


Figure A-2 — Construction of the codestream

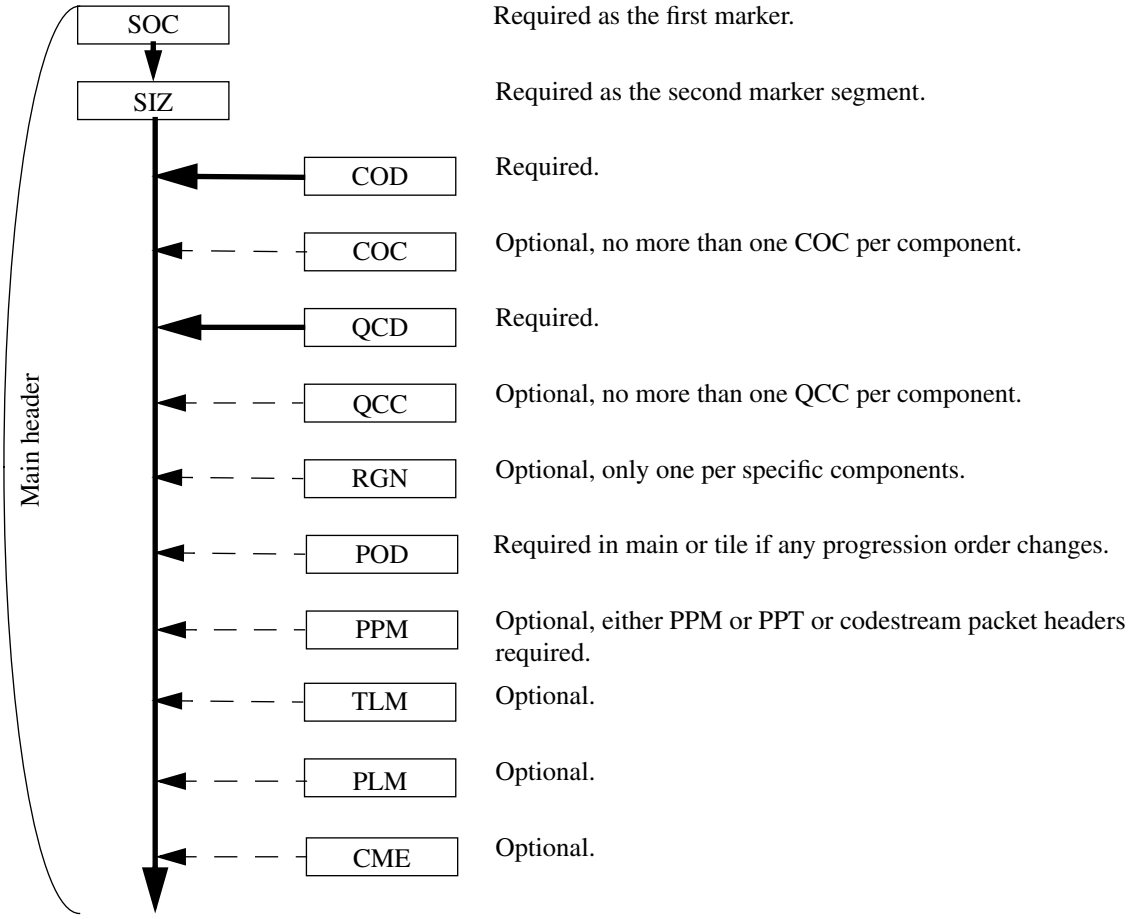


Figure A-3 — Construction of the main header

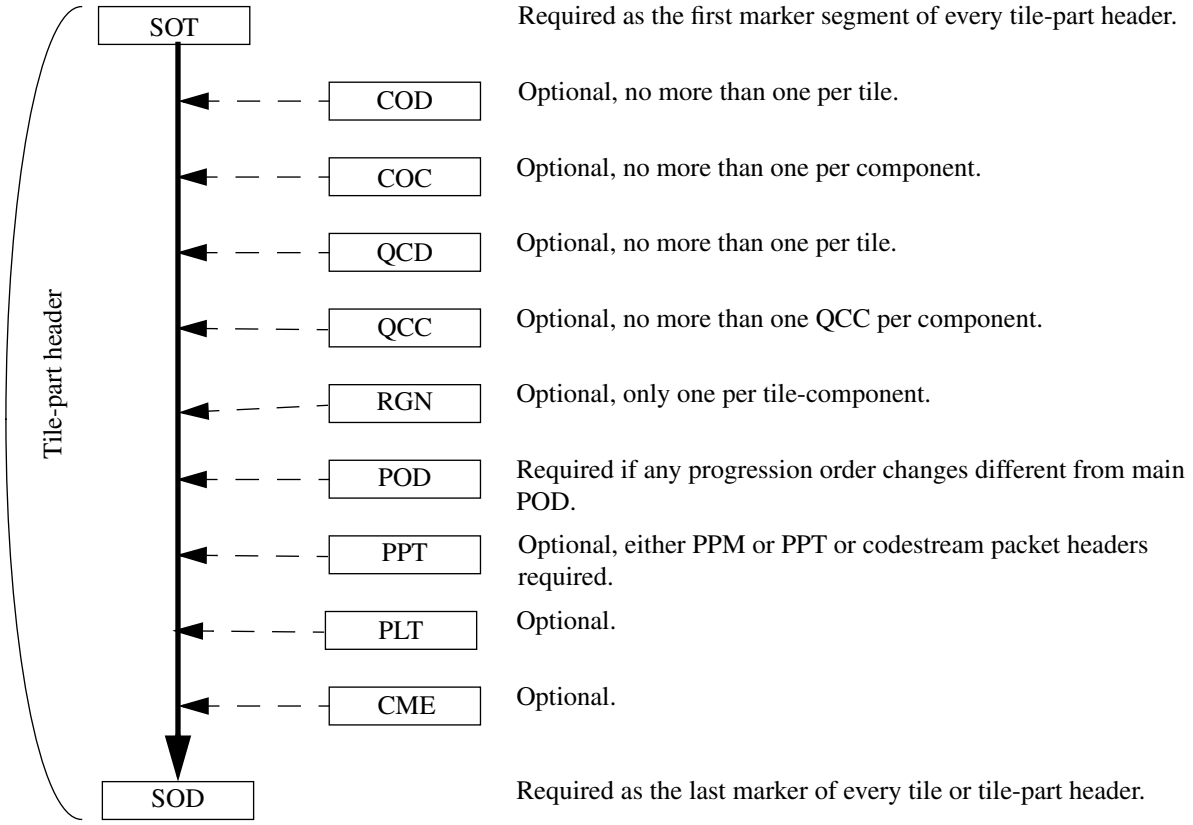


Figure A-4 — Construction of a tile-part header

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

For example, the COD marker segment is required in the main header. If all components in all the tiles are coded the same way, this is all that is required. If there is one component that is coded differently than the others (for example the luminance component of an image composed of luminance and 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 POD marker likewise may appear in the main header, and is used in all tiles, unless a different POD appears in the tile header.

A.4 Delimiting markers

The delimiting 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 (SOT and SOD). Each tile-part has one SOT and one SOD marker. The SOC, SOD, and EOC delimiting markers are 16 bits in length with no explicit length information.

A.4.1 Start of codestream (SOC)

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

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

Length: Fixed.

SOC: Marker value.

Table A-4 — Start of codestream parameter values

Parameter	Size (bits)	Values
SOC	16	0xFF4F

A.4.2 Start of tile-part (SOT)

Function: Marks the beginning of a tile-part and the index of its tile within a codestream. The tile-parts of a tile shall appear in order (see TPsot) in the codestream, but not necessarily consecutively.

Usage: 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.

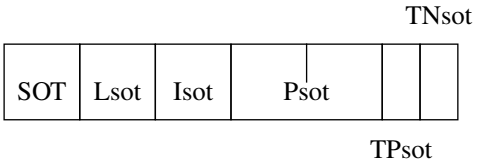


Figure A-5 — Start of tile-part syntax

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

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

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

Psot: Length, in bytes, from the beginning of the first byte of this SOT marker segment of the tile-part to the end of the data of that tile-part. Figure A-13 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 instance. If this is a tile-part, there is a specific order required for decoding tile-parts; this index then 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 defined 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 — 65 535
Psot	32	0 — $(2^{32}-1)$
TPsot	8	0 — 255
TNsot	8	0 — 255

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

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

Value	Number of tile-parts
1 — 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: Shall be the last marker in a tile-part header. 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 (see Annex D.4.2). There shall be at least one SOD in a codestream. There shall be one SOD per tile-part.

Length: Fixed.

SOD: Marker value

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

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

Length: Fixed.

EOC: Marker value

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.

Usage: 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.

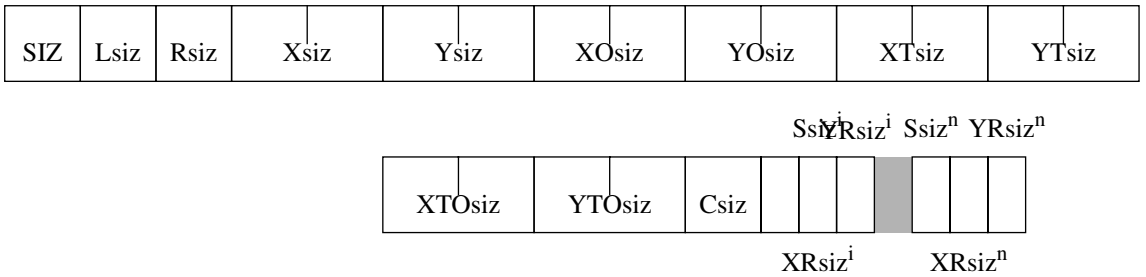


Figure A-6 — Image and tile size syntax

SIZ: Marker value. Table A-9 shows the size and parameter values for image and tile size.

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

Rsiz: Denotes capabilities of the codestream.

Xsiz: Width of the reference grid.

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. The precision is the precision of the component before the RCT or ICT is performed. (It is not necessarily the precision of the component plane coded in the file. The ICT or RCT can change the precision.) There is one occurrence of this parameter for each component. This parameter signals the component precision that is in the codestream. Only those bit-planes necessary need be extracted.

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	42 — 49 191
Rsiz	16	use Table A-10
Xsiz	32	1 — (2^{32} - 1)
Ysiz	32	1 — (2^{32} - 1)
XOsiz	32	0 — (2^{32} - 2)
YOsiz	32	0 — (2^{32} - 2)
XTsiz	32	1 — (2^{32} - 1)
YTsiz	32	1 — (2^{32} - 1)
XTOsiz	32	0 — (2^{32} - 2)
YTOsiz	32	0 — (2^{32} - 2)
Csiz	16	1 — 16 384
$Ssiz^{i}$	8	use Table A-11
$XRsiz^{i}$	8	1 — 255
$YRsiz^{i}$	8	1 — 255

Table A-10 — Capability Rsiz parameter

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

Table A-11 — Component Ssiz parameter

Values (bits) MSB LSB	Coefficient size
x000 0000 x010 0101	Components are value+1; from 1 bit deep through 38 bits deep respectively (including the sign bit, if appropriate)[a]
0xxx xxxx	Components are unsigned values
1xxx xxxx	Components are signed values

Table A-11 — Component Ssiz parameter

Values (bits) MSB LSB	Coefficient size
	All other values reserved.

a. The component precision is limited by the number of guard bits, quantization, growth of coefficients at each level of the transform, 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. If there are multiple tile-parts for a tile, then these marker segments shall be found only in the first tile-part (Tsot = 0).

A.6.1 Coding style default (COD)

Function: Describes the coding style, decomposition, 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) that are not described by COC marker segment. The parameter values can be overridden for an individual component by a COC marker segment in either the main or tile-part header.

Usage: Shall be one and only one in the main header. There may be at most one for all tile-part headers of a 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 (Tsot = 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.

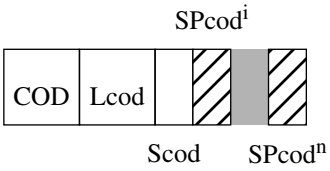


Figure A-7 — Coding style default syntax

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

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

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

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

Table A-12 — Coding style default parameter values

Parameter	Size (bits)	Values
COD	16	0xFF52
Lcod	16	12 — 65 535
Scod	8	Table A-13
SPcod^i	variable	Table A-13

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

Values (bits) MSB LSB	Coding style	SPcod usage
0000 00x0 0000 00x1	Entropy coder, without partitions (implies PPx = 2^{15} and PPy = 2^{15}) Entropy coder, with partitions	Table A-14
xxxx xx0x xxxx xx1x	No SOP marker segments used SOP marker segments may used	
xxxx x0xx xxxx x1xx	No EPH marker segments used EPH marker segments may used	
	All other values reserved	

Table A-14 — Coding style parameter values of the SPcod parameters

Parameters (in order)	Size (bits)	Values	Meaning of SPcod values
Decomposition levels	8	0 — 32	Number of decomposition levels, dyadic decomposition. (Zero implies no transform.)
Progression order	8	Table A-15	Progression order
Number of layers	16	0 — 65535	Number of layers
Code-block size width	8	Table A-16	Code-block width exponent value, xcb
Code-block size height	8	Table A-16	Code-block height exponent value, ycb
Code-block style	8	Table A-17	Style of the code-block coding passes
Transform	8	Table A-18	Wavelet transform used.
Multiple component transform	8	Table A-19	Multiple component transform usage
Packet partition size	variable	Table A-20	If partitions are not used, this parameter is not present. If partitions are used, this indicates partition size width and height. The first parameter (8 bits) corresponds to the LL sub-band. Each successive parameter corresponds to each successive decomposition level in order.

Table A-15 — Progression order for the SPcod, Ppod parameters

Values (bits) MSB LSB	Progression order
0000 0000	Layer-resolution-component-position progressive
0000 0001	Resolution-layer-component-position progressive
0000 0010	Resolution-position-component-layer progressive
0000 0011	Position-component-resolution-layer progressive

Table A-15 — Progression order for the SPcod, Ppod parameters

Values (bits) MSB LSB	Progression order
0000 0100	Component-position-resolution-layer progressive
	All other values reserved

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

Values (bits) MSB LSB	Code-block width and height
xxxx 0000 — xxxx 1000	Code-block width and height offset exponent value $xcb = 2^{value+2}$ or $ycb = 2^{value+\bar{2}}$. 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 to the xcb+ycb <= 12.
	All other values reserved

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

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

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

Values (bits) MSB LSB	Transform type
0000 0000	9-7 irreversible wavelet transform

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

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

Table A-19 — Multiple component transformation CSsiz parameter

Values (bits) MSB LSB	Multiple component transformation type
0000 0000	No multiple component transform specified. (A multiple component transform may be specified by the file format level.)
0000 0001	Reversible component transform on components 0, 1, 2 (see Annex G.2). Shall be used only with the 5-3 reversible wavelet transform.
0000 0010	Irreversible component transform on components 0, 1, 2 (see Annex G.2)Shall be used only with the 9-7 irreversible wavelet transform.
	All other values reserved

Table A-20 — Packet partition width and height for the SPcod parameters

Values (bits) MSB LSB	Packet partition size
xxxx 0000 — xxxx 1111	4 LSBs are the packet partition width exponent, $PPx = value$. Only the first value may equal zero.
0000 xxxx — 1111 xxxx	4 MSBs are the packet partition height exponent $PPy = value$. Only the first value may equal zero.

A.6.2 Coding style component (COC)

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

Usage: Optional in both the main and tile-part headers. No more than one per 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 (Tsot = 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.

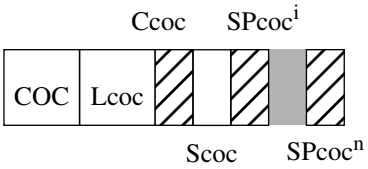


Figure A-8 — Coding style component syntax

COC: Marker value. Table A-21 shows the size and parameter values for coding styles.

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

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

Scoc: Coding style for this component. Table A-22 shows the value for each Scoc parameter.

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

Table A-21 — Coding style component parameter values

Parameter	Size (bits)	Values
COC	16	0xFF53
Lcoc	16	9 — 65 535
Ccoc	8 16	0 — 255; if Csiz < 257 0 — 16383; Csiz ≥ 257
Scoc	8	Table A-22
SPcoc^i	variable	Table A-22

Table A-22 — Coding style parameter values for the Scoc parameters

Values (bits) MSB LSB	Coding style	SPcoc usage
0000 0000	Entropy coder, PARTITION = 0	Table A-23
0000 0001	Entropy coder, PARTITION = 1	Table A-23
	All other values reserved	

Table A-23 — Coding style parameter values from SPcoc parameters

Parameters (in order)	Size (bits)	Values	Meaning of SPcoc values
Decomposition levels	8	0 — 32	Number of decomposition levels, dyadic decomposition. (Zero implies no transform.)
Code-block size width	8	Table A-16	Code-block width exponent value of the number 2, xcb
Code-block size height	8	Table A-16	Code-block height exponent value of the number 2, ycb
Code-block context	8	Table A-17	Style of the code-block coding passes
Transform	8	Table A-18	Wavelet transform used.
Packet partition size	variable	Table A-20	If PARTITION = 0, not present If PARTITION = 1, indicates partition size width and height First is LL, then repeated for every decomposition level

A.6.3 Region-of-interest (RGN)

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

Usage: May be used in main or tile-part header. 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 maker.

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.

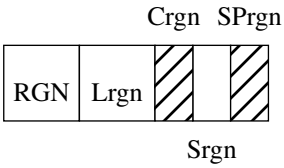


Figure A-9 — Coding style default syntax

RGN: Marker value. Table A-24 shows the size and parameter values for coding styles.

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

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

Srgn: ROI style for the current ROI. Table A-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 — 6
Crgn	8 16	0 — 255; if Csiz < 257 0 — 16383; Csiz ≥ 257
Srgn	8	Table A-25
SPrgn	variable	Table A-26

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

Values	ROI style (Srgn)	SPrgn usage
0	Implicit ROI (maximum shift)	Table A-26
	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 — 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: 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 (Tsot = 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.

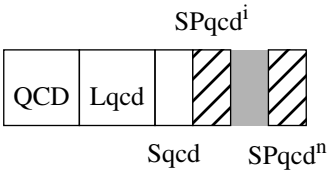


Figure A-10 — Quantization default syntax

QCD: Marker value. Table A-27 shows the size and parameter values for coding styles.

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

Sqcd: Quantization style for all components.

SPqcd^i:Quantization step size value for the ith sub-band in the defined order (see Annex B.6). The number of parameters is the same as, or larger than, the number of sub-bands in the tile-component with the greatest number of decomposition levels. If the number of parameters exceed number of sub-bands the last parameters in the series is ignored.

Table A-27 — Quantization default parameter values

Parameter	Size (bits)	Values
QCD	16	0xFF5C
Lqcd	16	4 — 197
Sqcd	8	Table A-28
SPqcd^i	variable	Table A-28

Table A-28 — Quantization default values for the Sqcd and Sqcc parameters

Values (bits) MSB LSB	Quantization style	SPqcx size (bits)	SPqcx usage
xxx0 0000	No quantization	8	Table A-29
xxx0 0001	Scalar implicit (values signalled for LL sub-band only)	16	Table A-30

Table A-28 — Quantization default values for the Sqcd and Sqcc parameters

Values (bits) MSB LSB	Quantization style	SPqcx size (bits)	SPqcx usage
xxx0 0010	Scalar explicit (values signalled for each sub-band)	16	Table A-30
000x xxxx — 111x xxxx	Number of guard bits 0 — 7		
	All other values reserved		

Table A-29 — Reversible step size values for the SPqcd and SPqcc parameters (5-3 transform only)

Values (bits) MSB LSB	Reversible step size values
xxx0 0000 — xxx1 1111	Exponent, ε_b, of the reversible dynamic range (signalled for each sub-band)
	All other values reserved

Table A-30 — Quantization values for scalar quantization for the SPqcd and SPqcc parameters (9-7 transform only)

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

A.6.5 Quantization component (QCC)

Function: Describes the quantization used for compressing a particular component

Usage: Optional in both the main and tile-part headers. No more than one per 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 (Tsot = 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.

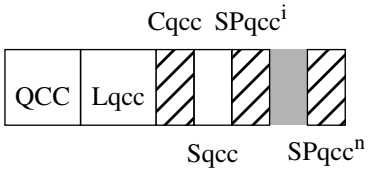


Figure A-11 — Quantization component syntax

QCC: Marker value. Table A-31 shows the size and parameter values for coding styles.

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

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

Sqcc: Quantization style for this component.

SPqcc^i:Quantization value for each sub-band in the defined order (see Annex B.6). If used in the main header, the number of parameters is greater than, or equal to, the greatest number of sub-bands of this component across all tiles in the image. If used in the tile header, the number of parameters is greater than, or equal to, the number of sub-bands of the current tile-component.

Table A-31 — Quantization component parameter values

Parameter	Size (bits)	Values
QCC	16	0xFF5D
Lqcc	16	5 — 199
Cqcc	8 16	0 — 255; if Csiz < 257 0 — 16383; Csiz ≥ 257
Sqcc	8	Table A-28
SPqcc^i	variable	Table A-28

A.6.6 Progression order change, default (POD)

Function: Describes the bounds and progression order for any progression order other than default in the codestream.

Usage: May be used in the main or tile-part header. At most one POD may appear in the main header. At most one POD may appear in a tile-part header. A POD appearing in a tile-part header overrides any POD in the main header, for that tile only. PODs appearing in tile-parts other than the first tile part may contain progression order only for packets contained in that tile-part.

This tag if present overrides the progression field of the COD marker segments in the main and tile headers.

Tile-part POD > Tile-part COD > Main POD > Main COD

Each set of starting and ending parameters must be disjoint from any previous set of starting and ending parameters. Further, for any packet with a given component, layer, resolution, and position, all packets with the same component, resolution, position and a lower layer must appear before the packet with the given values.

Length: Variable depending on the number of different progressions.

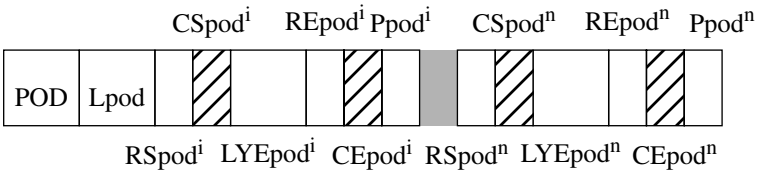


Figure A-12 — Progression order change, tile syntax

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

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

RSpod^i:Resolution index 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 parameters.

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

LYEpod^i:Layer index for the end of a progression. The layer index always starts at zero for every progression. Layers 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 parameters.

REpod^i:Resolution index 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 parameters.

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

Ppod: 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 parameters.

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

Parameter	Size (bits)	Values
POD	16	0xFF5F

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

Parameter	Size (bits)	Values
Lpod	16	9 — 65 535
$RSpod^i$	8	0 — 255
$CSpod^i$	8 16	0 — 255; if Csiz < 257 0 — 16383; Csiz ≥ 257
$LYEpod^i$	16	0 — 65535
$REpod^i$	8	0 — 255
$CEpod^i$	8 16	0 — 255; if Csiz < 257 0 — 16383; Csiz ≥ 257
$Ppod^i$	8	Table A-15

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 in the bit stream of the packets.

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

The TLM (Ptlm) or the SOT (Psot) parameters point from the beginning of the current tile-part's SOT marker segment to the end of the data in that tile-part. Because tile-parts are required to be a multiple of 8 bits, these values are always a byte length. Figure A-13 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 are lengths of each packet in the tile-part.

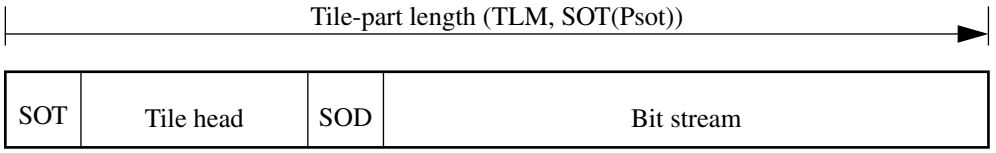


Figure A-13 — Coded tile-part lengths

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

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

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

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

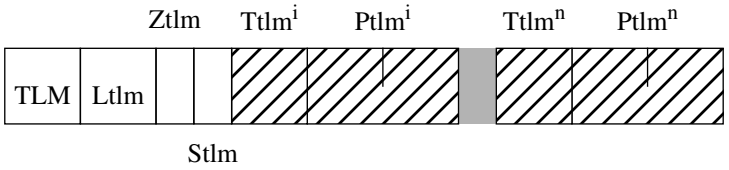


Figure A-14 — Tile-part length, main header syntax

TLM: Marker value. Table A-33 shows the size and values for the tile-part length main header parameters.

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

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

Stlm: Size of the Ttlm and Ptlm parameters.

Ttlm^i: Tile number of the ith tile-part. Either none or one value for every tile-part. The number of tile-parts can be derived from the length of this marker segment 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 data for that tile-part. One value for every tile-part. The number of tile-parts can be derived from the length of this marker segment.

Table A-33 — Tile-part length, main header parameter values

Parameter	Size (bits)	Values
TLM	16	0xFF55
Ltlm	16	6 — 65 535
Ztlm	8	0 — 255
Stlm	8	Table A-34
$Ttlm^i$	0 if ST = 0 8 if ST = 1 16 if ST = 2	tile in order 0 — 254 0 — 65 334
$Ptlm^i$	16 if SP = 0 32 if SP = 1	2 — 65 534 2 — (2^{32}-2)

Table A-34 — Size parameters for Stlm

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

A.7.2 Packet length, main header (PLM)

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

Usage: Can be used in the main header only. 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.

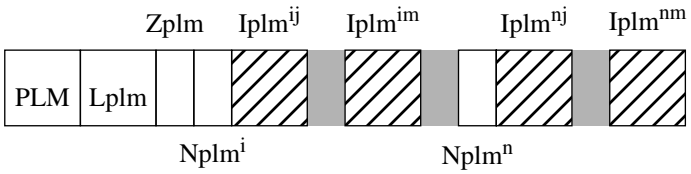


Figure A-15 — Packets length, main header syntax

PLM: Marker value. Table A-35 shows the size and values for the packets, main header parameters.

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

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

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

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

Table A-35 — Packets length, main header parameter values

Parameter	Size (bits)	Values
PLM	16	0xFF57
Lplm	16	5 — 65 535
Zplm	8	0 — 255
Nplm^{i}	8	0 — 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 — x111 1111	Last 7 bits of packet length, terminate number[a] Continue reading[b] 7 bits of packet length

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

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

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

Usage: 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.

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

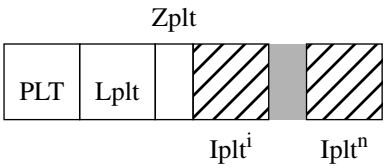


Figure A-16 — Packet length, tile-part header syntax

PLT: Marker value. Table A-37 shows the size and values for the packet parameters.

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

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

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

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

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

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

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

Usage: May be used in the main header for all tile-parts unless a PPT marker 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 are found in that marker segment. In this case, the PPT marker segment and packets distributed in the bit stream of the tile-parts are disallowed.

If there is no PPM marker segment then the packets can be distributed either in a PPT marker segment in the first tile-part (Tsot = 0) or distributed in the codestream as defined in Annex B.9. The packet headers shall not be in both a PPT marker segment and the codestream for the same tile. There may be multiple PPM marker segments in the main header.

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

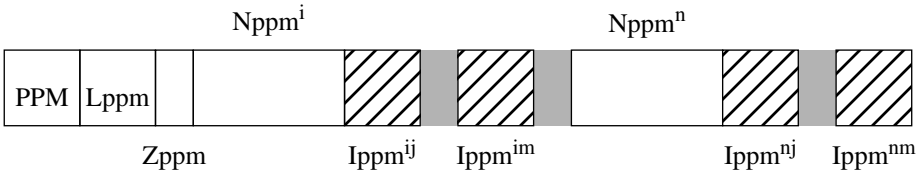


Figure A-17 — Packed packet headers, main header syntax

PPM: Marker value. Table A-38 shows the size and values for the parameters.

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

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

Nppm^{i}:Number of bytes of Iplm 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 component number, layer, and resolution are determined from the method of progression or the POD marker(s). The contents are exactly the packet header which would have been distributed in the bit stream as described in Annex B.8 packet header information. One range of values for each tile-part. One value for each packet in each tile-part.

Table A-38 — Packed packet headers, main header parameter values

Parameter	Size (bits)	Values
PPM	16	0xFF60
Lppm	16	6 — 65 535
Zppm	8	Table A-39
Nppm^{i}	32	0 - 65 535
Ippm^{ij}	variable	packet headers

Table A-39 — Index for the PPM marker segment parameters for Zppm

Values (bits) MSB LSB	Index size
0xxx xxxx 1xxx xxxx	Data in this marker segment starts with the next tile-part Data in this marker segment continues with the tile-part from the last
x000 0000 — x111 1111	Index of the marker segment from 0 to 127

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

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

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

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

If there is no PPM marker segment than the packets can be distributed either in a PPT marker segment in the first tile-part (Tsot = 0) or distributed in the codestream as defined in Annex B.9. The packet headers shall not be in both a PPT marker segment and the codestream for the same tile. 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 compression of the packet headers.

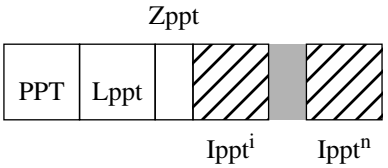


Figure A-18 — Packed packet headers, tile-part header syntax

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

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

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

Ippt^i: Packet header for every packet in order in the tile-part. The component number, layer, and resolution are determined from the method of progression or POD marker(s). The contents are exactly the packet header which would have been distributed in the bit stream as described Annex B.8 packet header information. One value for each packet in the tile.

Table A-40 — Packet header, tile-part headers parameter values

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

A.8 In bit stream marker segments

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

A.8.1 Start of packet (SOP)

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

Usage: Optional. Used in the bit stream in front of every packet. Shall only be used if indicated in the proper COD marker (see Annex A.6.1).

Length: Fixed 4 bytes.

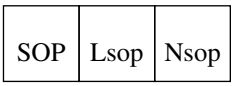


Figure A-19 — Start of packet syntax

SOP: Marker value. Table A-41 shows the size and values for start of tile-part.

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

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

Table A-41 — Start of packet parameter values

Parameter	Size (bits)	Values
SOP	16	0xFF91
Lsop	16	4
Nsop	16	0 — 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 headers in stream or in the PPM or PPT marker segments. This marker does not denote the beginning of packet data. If there is no packet header in stream, this marker shall not be used.

Usage: Optionally used in the bit stream or in the PPM or PPT marker segments. If there is no packet header in stream, this marker shall not be used. Shall only be used if indicated in the proper COD marker (see Annex A.6.1).

Length: Fixed.

EPH: Marker value

Table A-42 — End of packet header parameter values

Parameter	Size (bits)	Values
EPH	16	0xFF92

A.9 Informational markers

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

A.9.1 Comment and extension (CME)

Function: Allows unstructured data in the header.

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

Length: Variable depending on the length of the message.

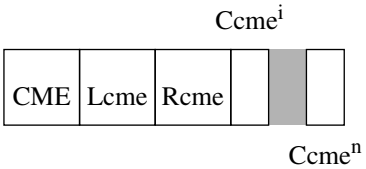


Figure A-20 — Coding style component syntax

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

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

Rcme: Registration value of the marker segment.

Ccme^i:Byte of unstructured data.

Table A-43 — Comment and extension parameter values

Parameter	Size (bits)	Values
CME	16	0xFF64
Lcme	16	5 — 65 535
Rcme	16	Table A-44
Ccme^i	8	0 — 255

Table A-44 — Registration values for the Rcme parameter

Values	Registration values
0	General use (binary values)
1	General use (ISO 8859-1 (latin-1) values)
2 — 65 534	Reserved for registration
65 535	Reserved for extension

Annex B

Data ordering

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

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

B.1 Image division into components

All components (and many other structures in this Annex) are defined with respect to a high resolution grid. The various parameters defining the reference grid appear in Figure B-1. The reference grid is a rectangular grid of data 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 grid point at location (XOsiz, YOsiz), and its lower right hand grid point at location (Xsiz-1, Ysiz-1).

The samples of component i are at integer multiples of (XRsiz(i), YRsiz(i)) on the reference grid. Only those samples which fall within the image area actually belong to the image component. Thus the samples of component i are mapped into the image component domain, as a rectangle having upper left hand sample with coordinates (x_0, y_0) and lower right hand sample with coordinates $(x_1\text{-}1, y_1\text{-}1)$ where

$$x_0 = \left\lceil \frac{XOsiz}{XRsiz(i)} \right\rceil \qquad x_1 = \left\lceil \frac{Xsiz}{XRsiz(i)} \right\rceil \qquad y_0 = \left\lceil \frac{YOsiz}{YRsiz(i)} \right\rceil \qquad y_1 = \left\lceil \frac{Ysiz}{YRsiz(i)} \right\rceil \tag{B.1}$$

Thus, the dimensions of component i are given by

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

The parameters, Xsiz, Ysiz, XOsiz, YOsiz, XRsiz(i) and YRsiz(i) are all defined in the SIZ marker (see Annex A.5.1).

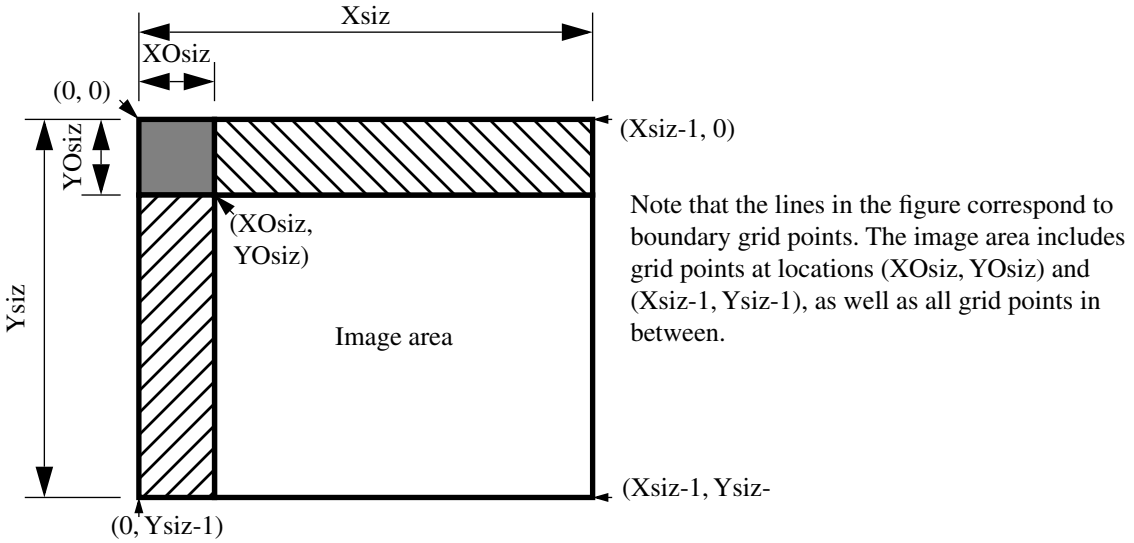


Figure B-1 — Reference grid diagram

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

B.2 Image 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 (see Annex A.5.1).

Every tile is XTsiz reference grid points wide and YTsiz reference grid points high. The top left corner on the first tile (tile 0) is offset from the top left corner of the reference grid by (XTOsiz, YTOsiz). The tiles are numbered in raster order. This is the tile number in the SOT marker from Annex A. Thus, the first tile's starting coordinates are (XTOsiz, YTOsiz). Figure B-2 shows this relationship.

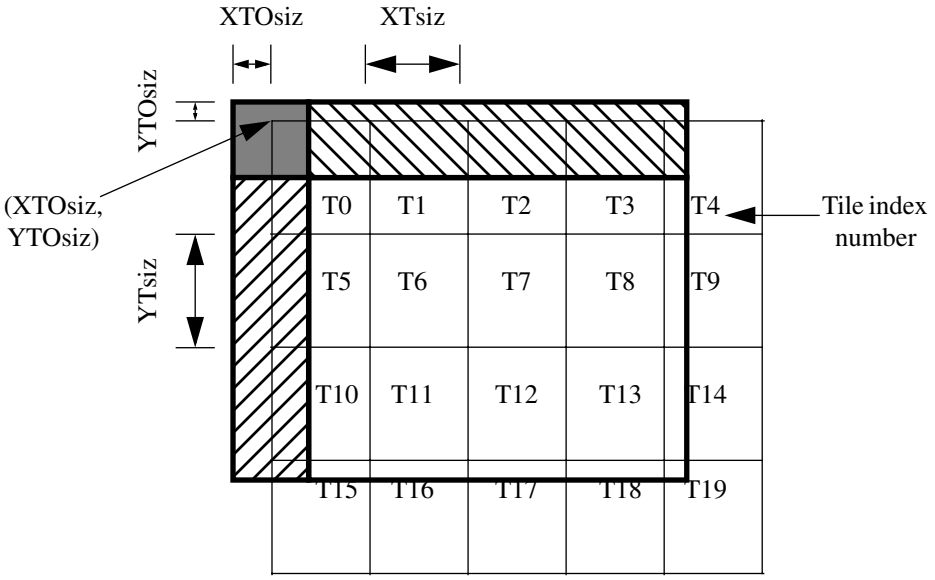


Figure B-2 — Tiling of the reference grid diagram

The tile grid offsets (XTOsiz and 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 number 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-2.

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

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

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

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

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

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 $ty_1(p, q)$ - 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

$$tcx_0 = \left\lceil \frac{tx_0}{XRsiz(i)} \right\rceil \qquad tcx_1 = \left\lceil \frac{tx_1}{XRsiz(i)} \right\rceil \qquad tcy_0 = \left\lceil \frac{ty_0}{YRsiz(i)} \right\rceil \qquad tcy_1 = \left\lceil \frac{ty_1}{YRsiz(i)} \right\rceil \tag{B.12}$$

so that the dimensions of the tile-component are

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

B.3 Example of the mapping of components to the reference grid

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 of with aspect ratio 4:3 can be

cropped from the center of the original image. Figure B-3 shows the reference grid and image areas along with the tiling structure that will be imposed in this example.

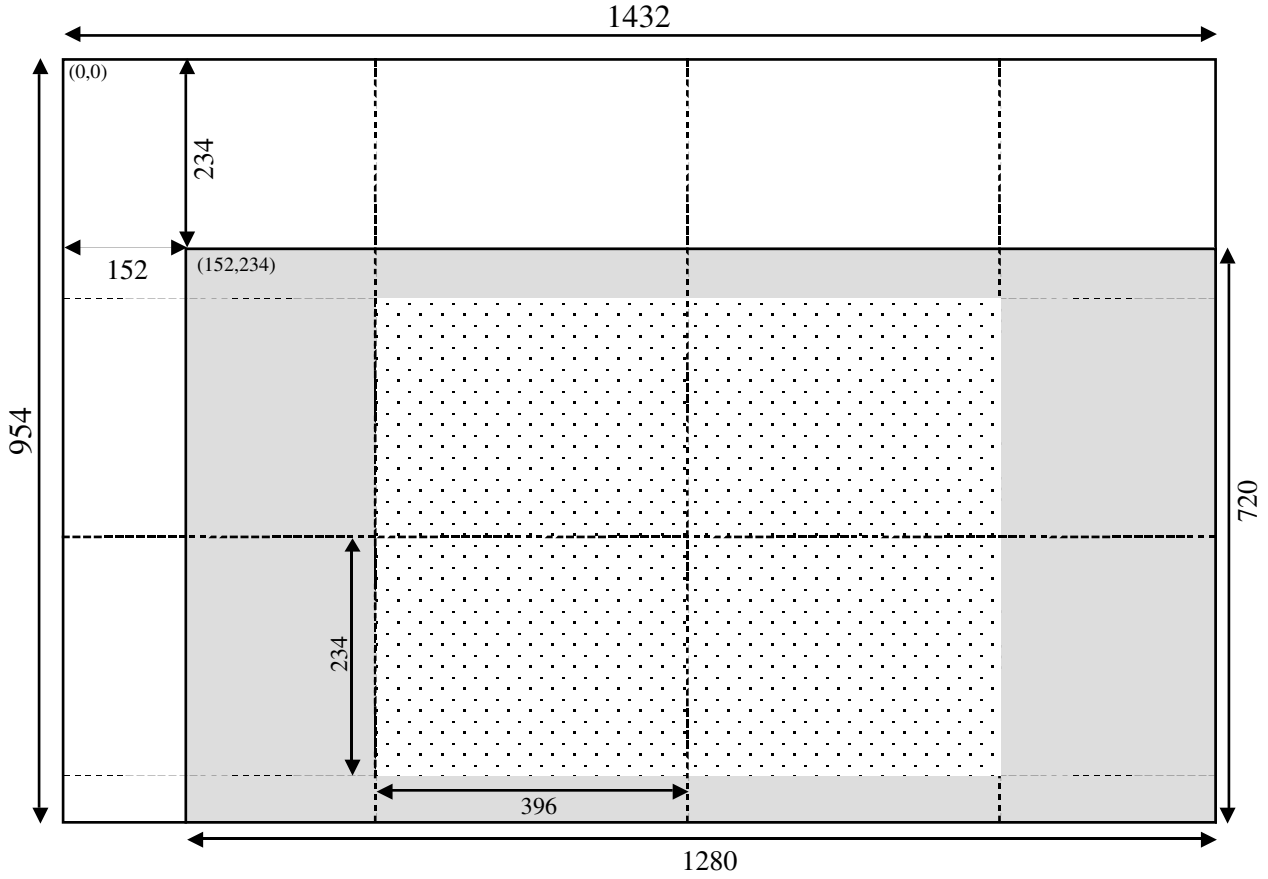


Figure B-3 — Reference grid example

Let the reference grid size (Xsiz, Ysiz) be (1432, 954). In this example, the image will contain two components (component numbers 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.13. 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) = (1 431 953). 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 left 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 plane. To do so, the values of tx_0, tx_1, ty_0, and ty_1 are determined from Equation B.7, Equation B.8, Equation B.9, and Equation 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 $ty_0(*,0:3)$ = {234,297,594,981}, and the values of ty_1 are $ty_1(*,0:3)$ = {297,594,891,954}.

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-4, while the same information is provided for component 1 in Figure B-5. 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(0), Yrsiz(0)) = (2,2). Notice that tiles (1,1) and (2,1) are both of size 198 x 148, while tiles (1,2) and (2,2) are both of size 198 x 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.

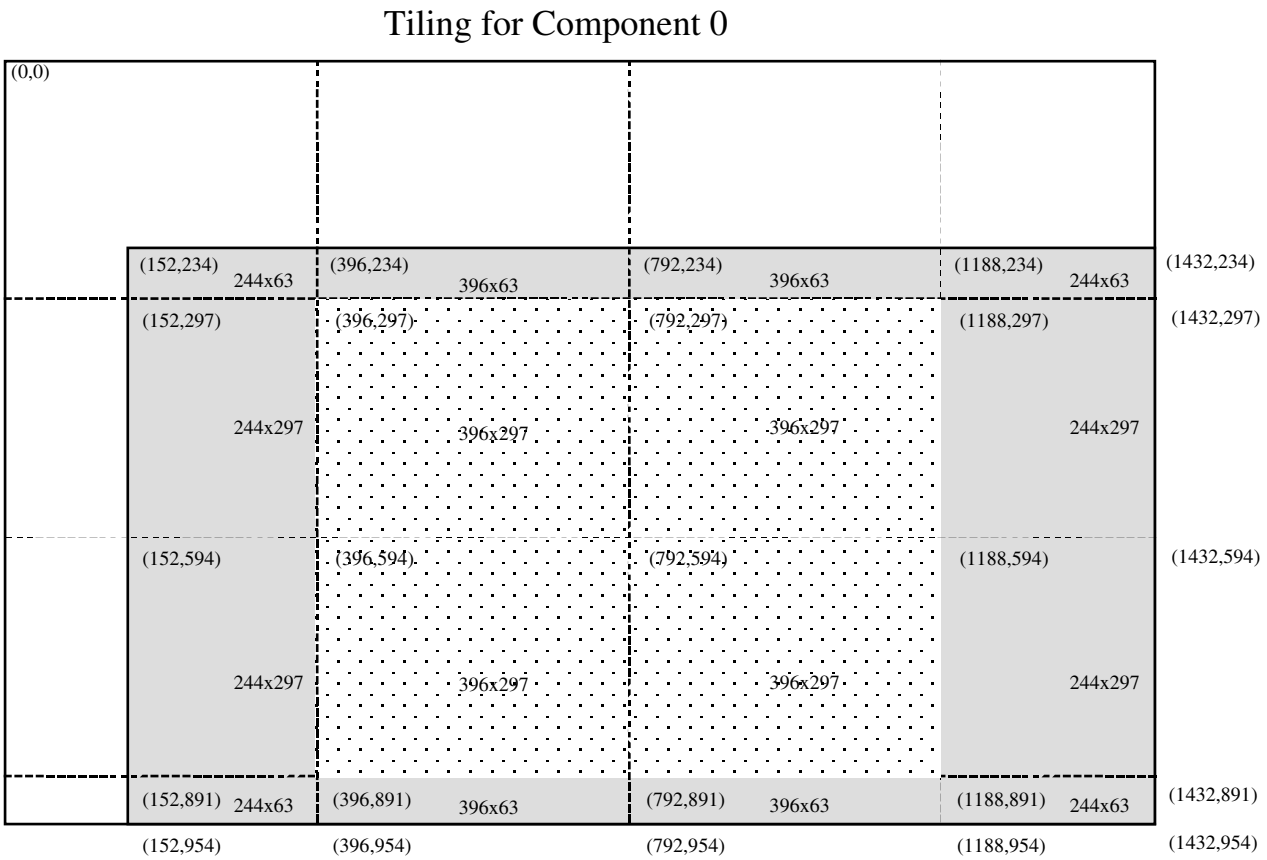


Figure B-4 — Example tile sizes and locations for component 0

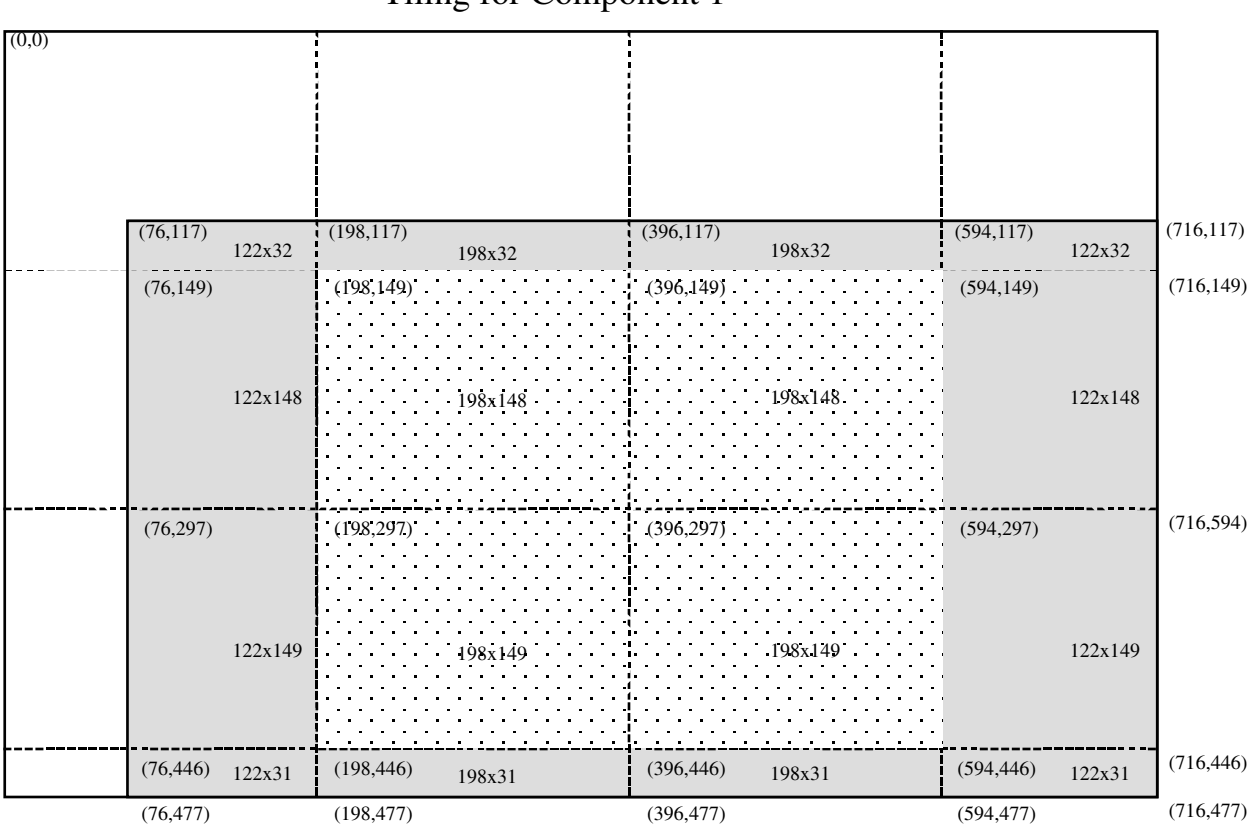


Figure B-5 — Example tile sizes and locations for component 1

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 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). Similarly, the image size must be adjusted to reflect the cropped size: (Xsiz', Ysiz') = (1188, 891). Finally, the tile offsets are no longer zero and instead must be set to (Xtsiz', Ytsiz') = (396,297).

B.4 Tile-component division into resolutions and sub-bands

Each image component is wavelet transformed with N_L decomposition levels as explained in Annex F. As a result, the component is available at N_L+1 distinct resolutions, denoted $r = 0,1,...,N_L$. The lowest resolution, $r = 0$, is represented by the N_LLL band. In general, resolution r is obtained by discarding sub-bands nHH, nHL, nLH for $n = 1$ through N_L-r and reconstructing the image component from the remaining sub-bands.

The tile coordinates are mapped into the image domain at any particular resolution, r, yielding upper left hand sample coordinates, (trx_0, try_0) and lower right hand sample coordinates, (trx_1-1, try_1-1), where

$$trx_0 = \left\lceil \frac{tcx_0}{2^{N_L - r}} \right\rceil \qquad try_0 = \left\lceil \frac{tcy_0}{2^{N_L - r}} \right\rceil \qquad trx_1 = \left\lceil \frac{tcx_1}{2^{N_L - r}} \right\rceil \qquad try_1 = \left\lceil \frac{tcy_1}{2^{N_L - r}} \right\rceil \qquad \text{B.14}$$

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

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

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 (horizontally high-pass)	1	0
n_bLH (vertically high-pass)	0	1
n_bHH	1	1

B.5 Division of resolutions into precincts

Consider a particular tile, component, and resolution, 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-6 shows the partitioning of this tile-component resolution into precincts. The precinct partition is anchored at location (0, 0), so that the upper left hand

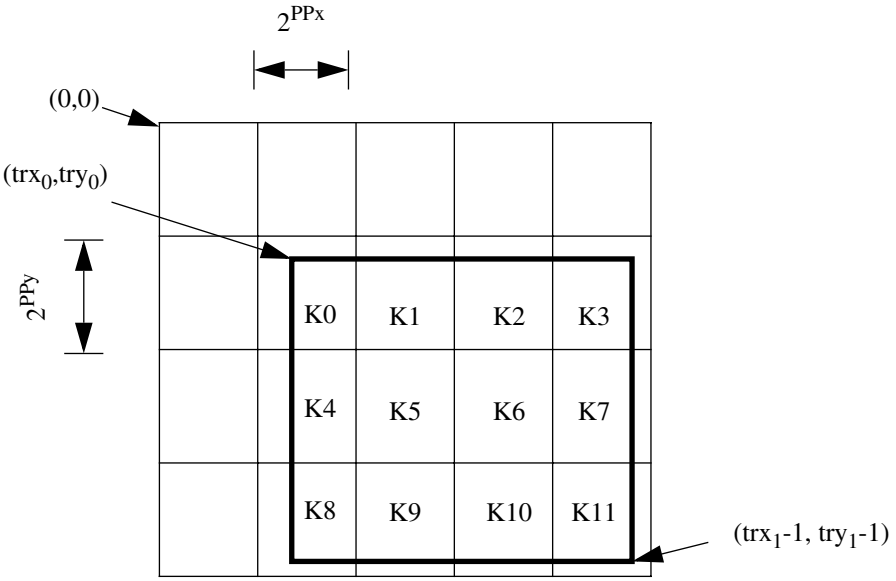


Figure B-6 — Precinct partition

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 markers (see Annex A.6.1 and Annex A.6.2). PPx and PPy may be different for each tile, component and resolution.

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

$$numkiwiswide = \left\lceil \frac{trx_1}{2^{PPx}} \right\rceil - \left\lfloor \frac{trx_0}{2^{PPx}} \right\rfloor \qquad numkiwishigh = \left\lceil \frac{try_1}{2^{PPy}} \right\rceil - \left\lfloor \frac{try_0}{2^{PPy}} \right\rfloor \tag{B.16}$$

The precinct index runs from 0 to *numprecincts* - 1 where *numprecincts* = *numprecinctswide* * *numprecincts* high in raster order (see Figure B-6). This is used in determining the order of appearance, in the codestream, of packets corresponding to each precinct, as explained in Annex B.11.

It can happen that a precinct is empty, meaning that no sub-band samples from the relevant resolution actually contribute to the precinct. When this happens every packet corresponding to that precinct must still appear in the codestream (see Annex B.8).

B.6 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 element of the partition is determined from two parameters, xcb and ycb, which are signalled in the COD or COC markers (see Annex A.6.1 and Annex A.6.2) and is the same for all sub-bands in the tile-component, at the same resolution, *r*. Specifically, the code-block size for the sub-bands is determined as $2^{xcb'}$ by $2^{ycb'}$ where

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

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 partition size and the nominal code-block size, whose parameters, xcb and ycb, are identical for all sub-bands in the tile-component. Like the precinct partition, the code-block partition is anchored at (0,0), as illustrated in Figure B-7. Thus, all first rows of code-blocks in the partition are located at $y = m2^{ycb'}$ and all first columns of code-blocks are located at $x = n2^{xcb'}$.

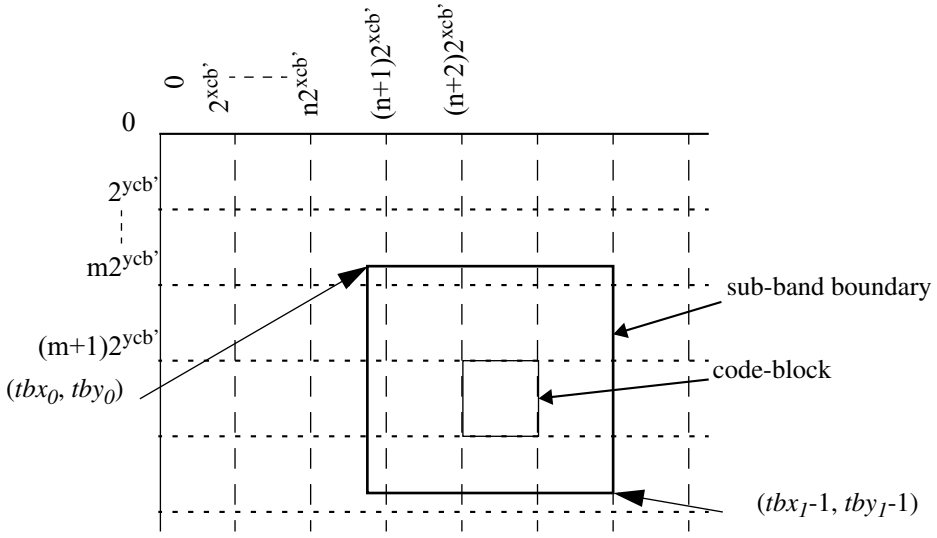


Figure B-7 — Code-blocks in sub-band *b*

NOTE — Code-blocks in the partition may extend beyond the boundaries of the sub-band data. When this happens, only the samples 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 lines of sub-band samples in the code-block or as many of such lines as are present.

B.7 Layers

The coded 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 signaled in the COD marker (see Annex A.6.1).

Each layer successively and monotonically improves the image quality, so that the decoder shall be able to decode the code-block contributions contained in each layer in sequence. For a given code-block, the first coding pass in layer n is the coding pass immediately following the last coding pass for the code-block in layer *n*-1, if any.

Layers are numbered from 0 to L-1, where L is the number of layers in the tile.

B.8 Packets

The data representing a specific tile, layer, component, resolution 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 Annex F.2.1, resolution $r = 0$ contains the sub-band samples from the N_LLL band, where N_L is the number of decomposition levels. Each subsequent resolution, $r > 0$, contains the sub-band samples from the nHL, nLH, and nHH sub-bands, as defined in Annex F, where $n = N_L - r + 1$. There are N_L+1 resolutions for a decomposition with N_L levels.

The 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 Annex F.2.1. Within each sub-band, the code-block contributions appear in raster order, confined to the bounds established by the relevant precinct. It is understood that resolution $r = 0$ contains only the LL band and resolutions $r > 0$ 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.

Packet data is introduced by a packet header whose syntax is described in Annex B.9, and 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.

It can happen that a precinct contains no code-blocks from any of the sub-bands at some resolution. When this happens, all packets corresponding to that precinct must appear in the codestream as empty packets, in accordance with the packet header syntax described in Annex B.9.

B.9 Packet header information coding

The packets have headers with the following information:

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

Two items in the header are coded with a scheme called tag trees described below. The data bits of the packet header are packed into a whole number of bytes with the bit stuffing routine described in Annex B.9.1.

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 Annex A.7.4). If the PPM is not used, then a PPT marker segment may be used. In this case, all of the packet headers in that tile are relocated to the first tile-part header (see Annex A.7.5).

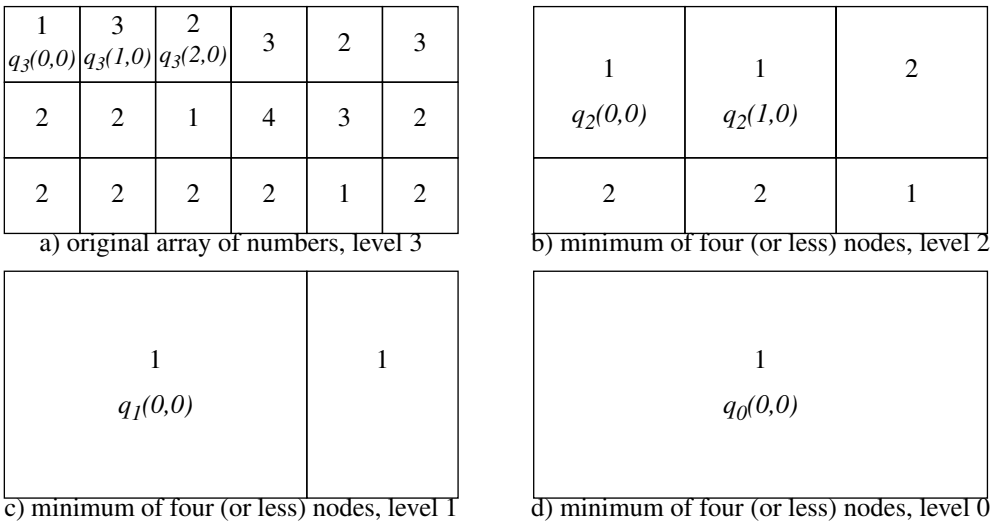


Figure B-8 — Example of a tag tree representation

B.9.1 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 one zero bit stuffed after a byte with 0xFF must be included even if the 0xFF would otherwise have been the last byte).

B.9.2 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-8 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; it contains the top node.

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 answered when adjacent code-blocks and/or layers were coded. This information is not coded again. Therefore, there is a causality to the information in packet headers.

NOTE — For example, in Figure B-8, the coding for the number at $q_3(0,0)$ would be 01111. The two bits, 01, imply that the top node at $q_0(0,0)$ is greater than zero and is, in fact one. The third bit, 1, implies that the node at $q_1(0,0)$ is also one. The fourth bit, 1, implies that the node at $q_2(0,0)$ is also one. And the final bit, 1, implies that the target node at $q_3(0,0)$ is also one. To decode the next node $q_3(1,0)$ the nodes at $q_0(0,0)$, $q_1(0,0)$, and $q_2(0,0)$ are already known. Thus, the bits coded are 001, the zero says that node at $q_3(1,0)$ is greater than 1, the second zero says it is greater than 2, and the one bit implies that the value is 3. Now that $q_3(0,0)$ 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 Table B-8a.

B.9.3 Zero length packet

The first bit in the packet header denotes whether the packet has a length of zero (not present). 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.

B.9.4 Code-block inclusion

Information concerning whether or not 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 layer, 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. The values in this tag tree are the number of the layer in which the code-block is first included. Note that only the bits needed for determining whether the 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.9.5 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.3. 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.9.6 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-2. Note that this table provides for the possibility of signalling up to 164 coding passes.

Table B-2 — Codewords for the number of coding passes for each code-block

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

NOTE — Since the value of M_b is limited to a maximum value of 38 by the constraints imposed by the syntax of the COD and COC markers (see Annex A.6.1 and Annex A.6.2), it is not possible for more than 109 coding passes to be employed by the block coding algorithm described in Annex D.

B.9.7 Length of the 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 terminate in a 0xFF. This is, in fact, not a burdensome requirement, since 0xFFs are always synthesized as necessary by the block decoder described in Annex C. 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 terminating with an 0xFF.

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 Table D-8 and Table D-9.

B.9.7.1 Single codeword segment

The number of bits used to signal the number of bytes contributed to a packet by a code-block is given by

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

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

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

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

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

B.9.7.2 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 Table D-8 and Table D-9. T is augmented with the final coding pass included in the packet. Let $n_1 < ... < n_K$ be the indices in T. K lengths are signaled consecutively with each length using the mechanism described in Annex B.9.7.1. 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 up to 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.

B.9.8 Order of information within packet header

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

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 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
        length of code-block contribution

The packet header may be immediately followed by the two-byte EPH marker as described in Annex A.8.2. In this case, the EPH marker must 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.

Figure B-9 shows a brief example. This is 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-3 shows the resulting bit stream (in part) from this information.

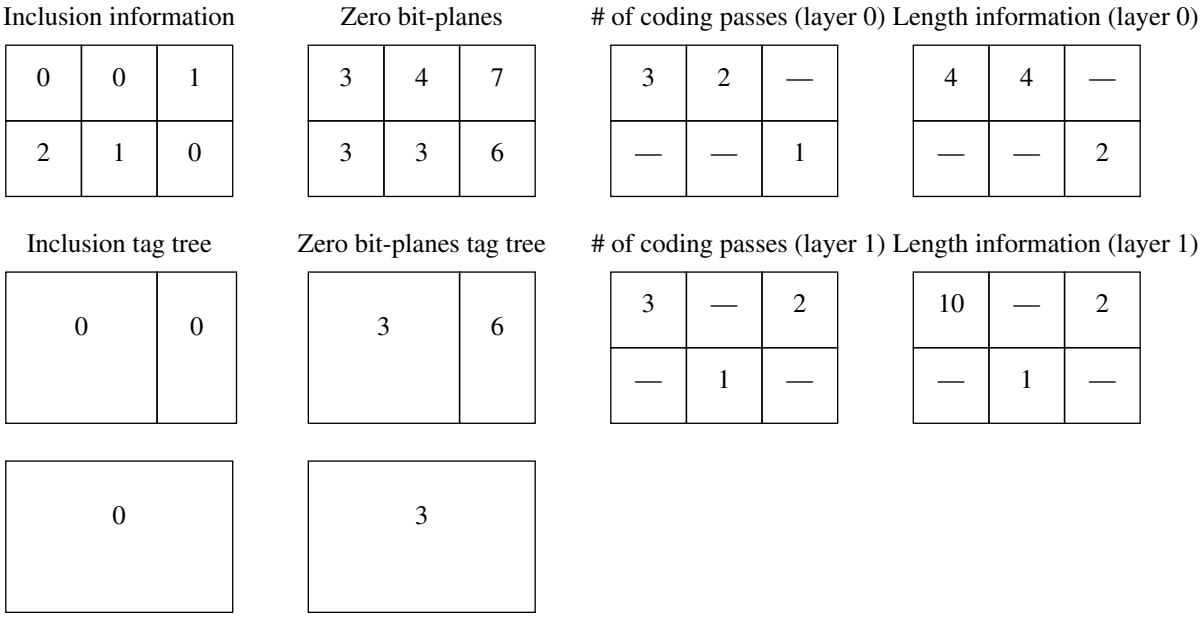


Figure B-9 — Example of the information known to the encoder

Table B-3 — Example packet header bit stream

Bit stream (in order)	Derived meaning
1	Packet non-zero in length
111	Block 0,0 included for the first time
000111	Block 0,0 insignificant for 3 bit-planes
1100	Block 0,0 has 3 coding passes included
0	Block 0,0 length indicator is unchanged

Table B-3 — Example packet header bit stream

Bit stream (in order)	Derived meaning
0100	Block 0,0 has 4 bytes 4 bits are used, 3 + floor($\log_2$ 3)
1	Block 1,0 included for the first time
01	Block 1,0 insignificant for 4 bit-planes
10	Block 1,0 has 2 coding passes included
10	Block 1,0 length indicator is increased by 1 bit (3 to 4)
00100	Block 1,0 has 4 bytes 5 bits are used 4 + floor($\log_2$ 2) (Note that while this is a legitimate entry, it is not minimal in code length.)
0	Block 2,0 not yet included
0	Block 0,1 not yet included
0	Block 1,1 not yet included
1	Block 2,1 included for the first time
00011	Block 2,1 insignificant for 6 bit-planes
0	Block 2,1 has 1 coding passes included
0	Block 2,1 length information is unchanged (3 bits)
010	Block 2,1 has 2 bytes 3 + $\log_2$ 1 bits used
•••	Packet header data for the other sub-bands, coded packet data

Packet for the next layer

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

Table B-3 — Example packet header bit stream

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

B.10 Tile Data and Tile-Parts

Each tile is represented by a sequence of packets. The order in which these packets appear within the tile is defined in Annex B.11. Note that it is possible for a tile to contain no packets whatsoever, 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. In this way, any number of whole packets may be dropped and the final packet appearing in the tile may be partially truncated.

The sequence of packets representing any particular tile may be divided into contiguous segments known as tile-parts. Each tile must contain at least one tile-part. The divisions between tile-parts must occur at packet boundaries. Each packet in any given tile-part is prepended with an SOP marker, if and only if SOP markers are to be used for that tile-part as signalled by COD markers, described in Annex A.6.1. If the packet headers are moved to a PPM or PPT marker, then the SOP marker appears immediately before the packet body in the tile-part data portion. Otherwise, it appears immediately before the packet header, again in the tile-part data portion.

While tiles are coherent geometric areas on the image, 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 (see Annex A.4.2), containing the index of the tile to which the tile-part belongs.

B.11 Progression Order

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

B.11.1 Progression order determination

The COD markers signal which of the five progression orders are used (see Annex A.6.1). The progression order can also be overridden with the POD marker (see Annex A.6.6). For each of the possible progression orders the mechanism to determine the order in which packets are included is described below.

B.11.1.1 Layer-resolution-component-position progressive

Layer-resolution-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, 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.

B.11.1.2 Resolution-layer-component-position progressive

Resolution-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, r, layer, l, and precinct, k.

B.11.1.3 Resolution-position-component-layer progressive

Resolution-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 ($y = ty_0$ or y divisible by $YRsiz(i) \cdot 2^{PPy(r,i) + N_L - r}$)
          if ($x = tx_0$ or x divisible by $XRsiz(i) \cdot 2^{PPx(r,i) + N_L - r}$)
            for the next precinct, k, in the sequence shown in Figure B-6
              for each l = 0, ... , L-1
                packet for component, i, resolution, r, layer, l, and precinct, k.

In the above, k can be obtained from:

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

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

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 precinct partitions, mapped into the reference grid. This note also applies to the loops given for the following two progressions.

B.11.1.4 Position-component-resolution-layer progressive

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

for each $y = ty_0, \ldots, ty_1\text{-}1$,
  for each $x = tx_0, \ldots, tx_1\text{-}1$,
    for each $i = 0, \ldots,$ Csiz-1
      for each $r = 0, \ldots, N_L$ where N_L is the number of decomposition levels for component i,
        if ($y = ty_0$ or y divisible by $YRsiz(i) \cdot 2^{PPy(r,i)+N_L-r}$)
          if ($x = tx_0$ or x divisible by $XRsiz(i) \cdot 2^{PPx(r,i)+N_L-r}$)
            for the next precinct, k, in the sequence shown in Figure B-6
              for each $l = 0, \ldots,$ L-1
                packet for component, i, resolution, 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.

B.11.1.5 Component-position-resolution-layer progressive

Component-position-resolution-layer progression is defined as the interleaving of the packets in the following order:

for each $i = 0, \ldots,$ Csiz-1
  for each $y = ty_0, \ldots, ty_1\text{-}1$,
    for each $x = tx_0, \ldots, tx_1\text{-}1$,
      for each $r = 0, \ldots, N_L$ where N_L is the number of decomposition levels for component i,
        if ($y = ty_0$ or y divisible by $YRsiz(i) \cdot 2^{PPy(r,i)+N_L-r}$)
          if ($x = tx_0$ or x divisible by $XRsiz(i) \cdot 2^{PPx(r,i)+N_L-r}$)
            for the next precinct, k, in the sequence shown in Figure B-6
              for each $l = 0, \ldots,$ L-1
                packet for component, i, resolution, r, layer, l, and precinct, k.

In the above, k can be obtained from Equation B.20.

B.11.2 Progression order default

The progression order and extent of progression in a tile is affected if a POD marker segment is present in either the main or tile header (see Annex A.6.6).

If a POD marker segment is present, then the progression loops in Annex B.11.1 go from

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

These ranges apply to the progression order provided in the COD marker. A new progression order is specified by Ppod to be used in place of that given in the COD marker, outside the ranges given above. The POD allows this new progression to be further limited by subsequent start and end points, CSpod, CEpod, RSpod, REpod and LEpod, with a new default progression order, Ppod, to be applied outside those limits. This process may be continued as often as desired by signalling successive start and end points and new default progression orders.

Although no restriction is placed on the allowable values for CSpod, CEpod, RSpod and REpod, in the event that the appropriately modified progression order loops from Annex B.11.1 identify packets with layer, component or resolution indices outside the available range, the relevant position in the packet sequence is understood to be skipped.

Likewise, in the event that the appropriately modified progression order loops from Annex B.11.1 identify packets which have been previously included, the relevant position in the packet sequence is understood to be skipped.

Figure B-10 shows an example of two progression loops for a single component image. First packets are sent in resolution-layer-component-position progression until the box labeled “First” in the figure is complete; then packets are sent in layer-resolution-component-position progression for the layers of all resolutions which were not previously sent.

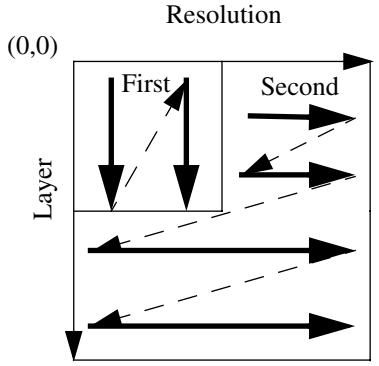


Figure B-10 — Example of progression order change in two dimensions

Annex C

Arithmetic entropy coding

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

This annex defines the lossless arithmetic entropy coding. This annex is compatible with the arithmetic coder defined in ITU-T Rec.T.88 | ISO/IEC 14492.

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.

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 data (CD) output. Both D and CX are provided by the model unit (not shown). CX selects the probability estimate to use during the coding of D. In this International Standard, CX is a label for a context.

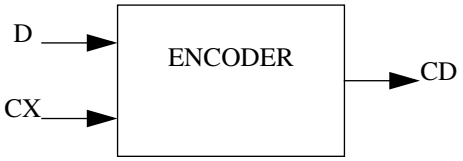


Figure C-1 — Arithmetic encoder inputs and outputs

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 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 $0{,}75 \le A < 1{,}5$ by doubling it whenever the integer value falls below 0x8000.

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

Keeping A in the range $0{,}75 \leq A < 1{,}5$ allows a simple arithmetic approximation to be used in the interval subdivision. The interval is A and the current estimate of the LPS probability is Qe, a precise calculation of the sub-intervals would require:

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

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

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

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

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

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 and 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 data are output when no longer modifiable. When all of the CX and D pairs have been read (Finished?), FLUSH sets the contents of the C-register to as many 1-bits as possible and then outputs the final bytes. FLUSH also terminates the encoding and generates the required terminating marker.

NOTE — While FLUSH is required in ITU-T Rec.T.88 | ISO/IEC 14492 it is informative in this specification. Other methods, such as that defined in Annex D.4.2, are acceptable.

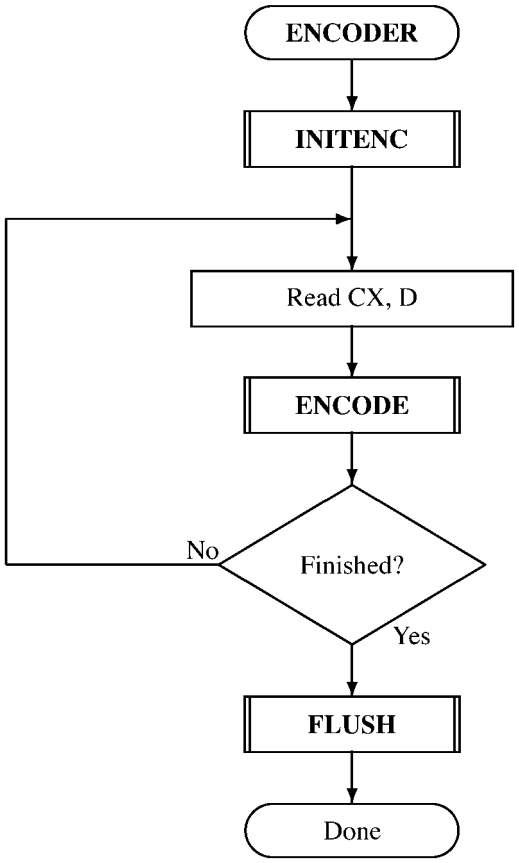


Figure C-2 — Encoder for the MQ-coder

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	aaaa aaaa	aaaa aaaa

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

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

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.

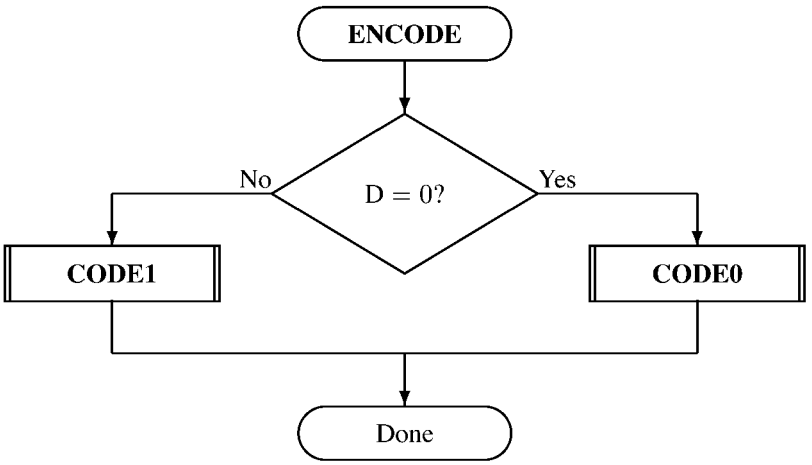


Figure C-3 — ENCODE procedure

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 Figure C-4 and Figure 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.

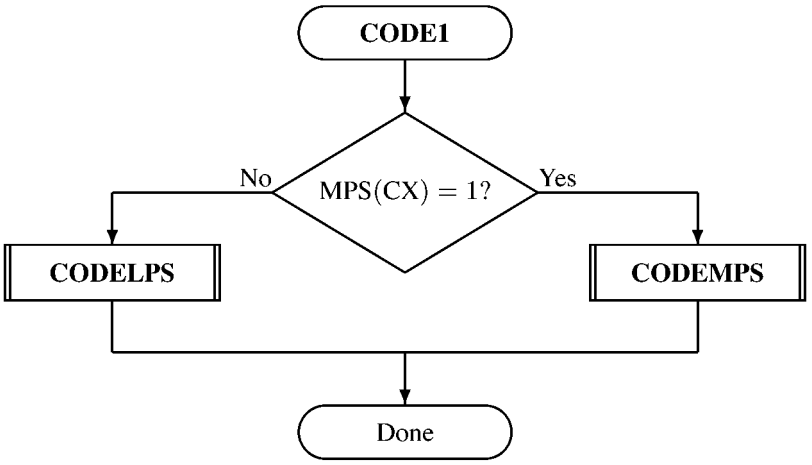


Figure C-4 — CODE1 procedure

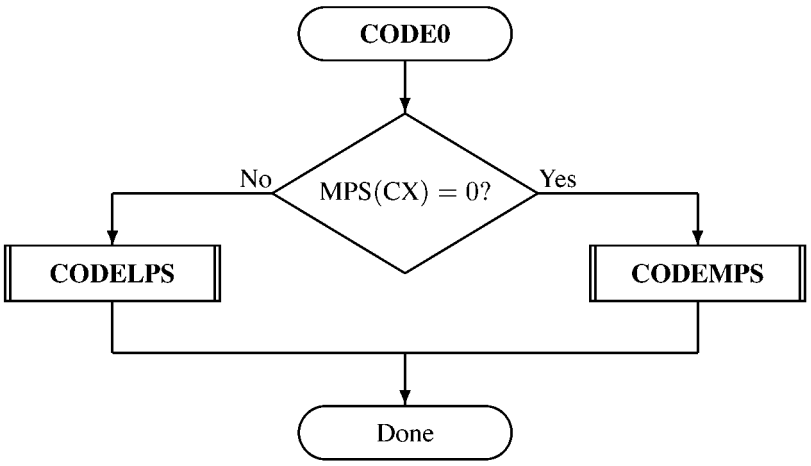


Figure C-5 — CODE0 procedure

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

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.

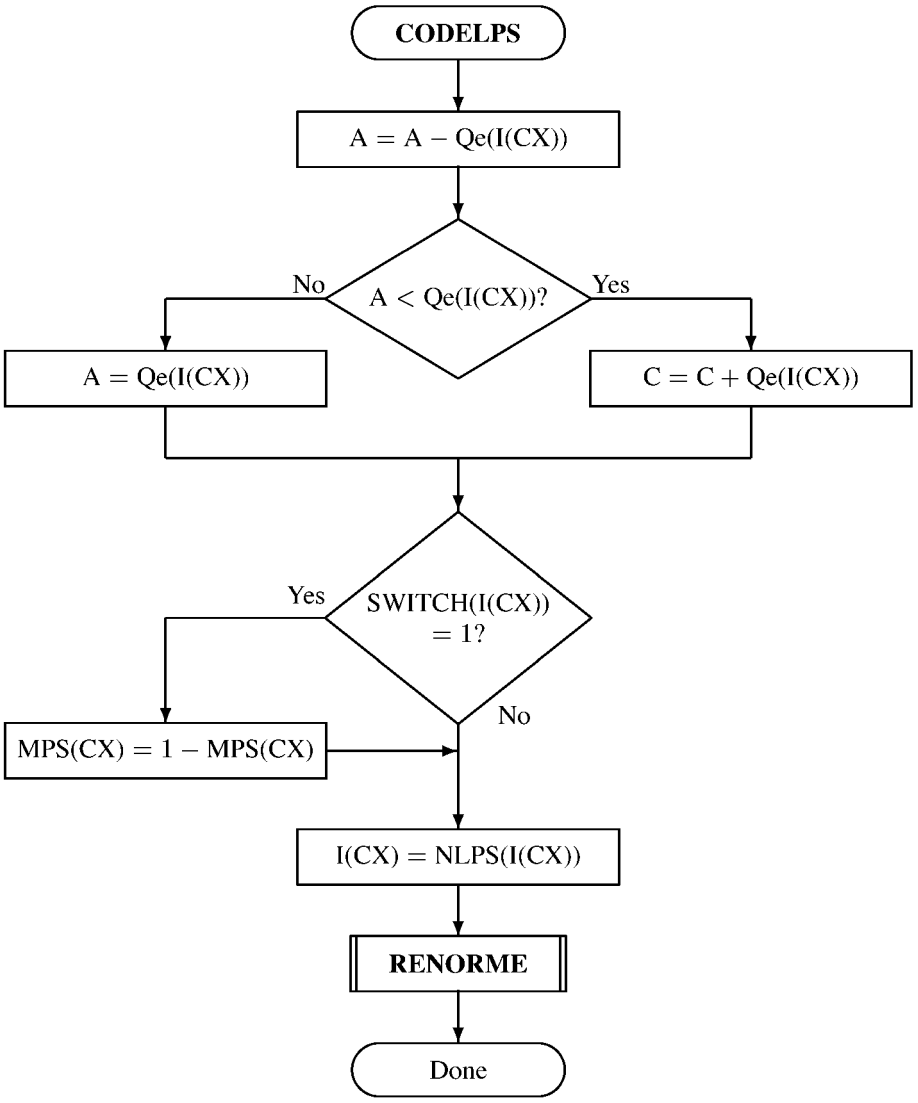


Figure C-6 — CODELPS procedure with conditional MPS/LPS exchange

Table C-2 — Qe values and probability estimation process

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

Table C-2 — Qe values and probability estimation process

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
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
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

Table C-2 — Qe values and probability estimation process

Index	Qe_Value			NMPS	NLPS	SWITCH
	(hexadecimal)	(binary)	(decimal)			
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

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

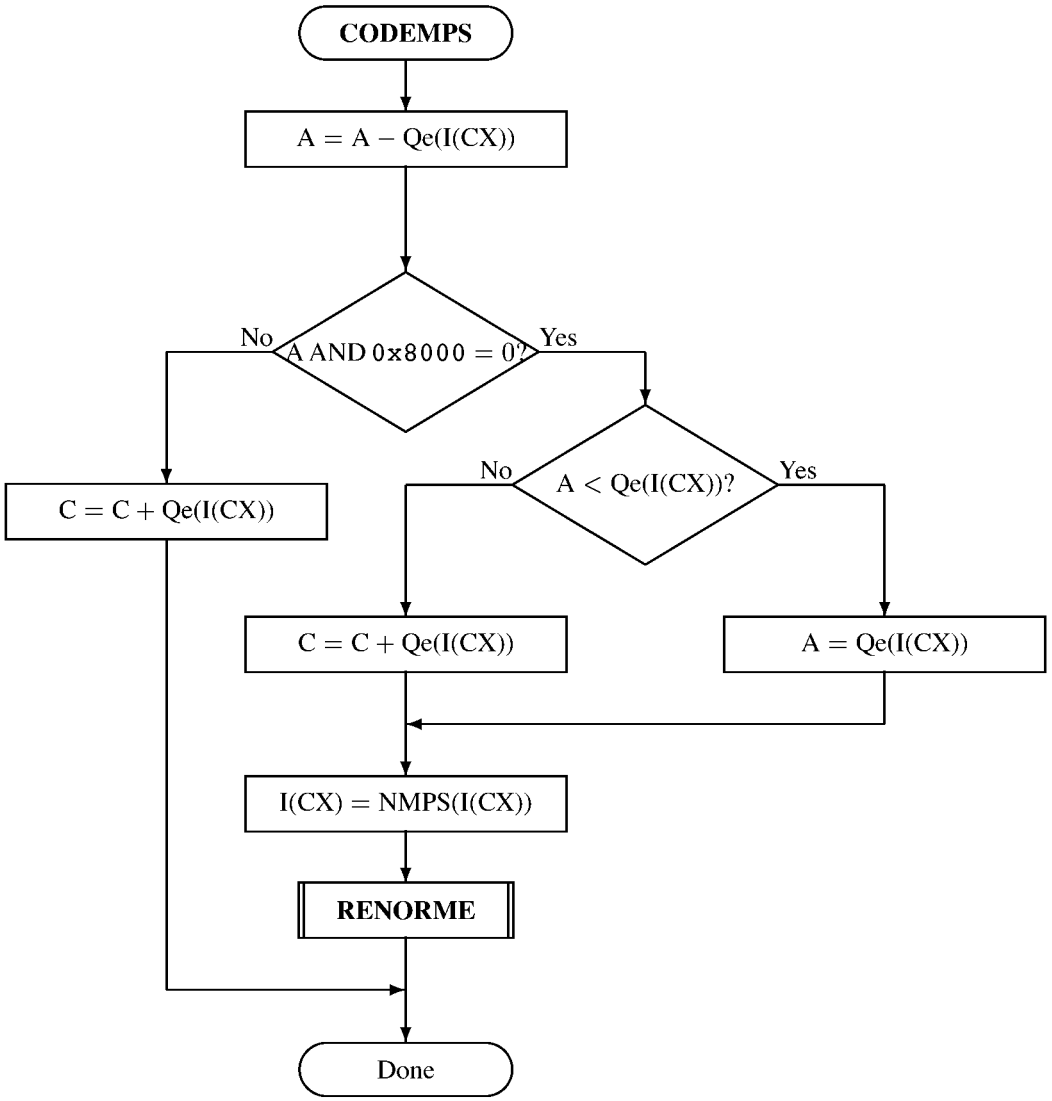


Figure C-7 — CODEMPS procedure with conditional MPS/LPS exchange

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

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

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

The final index state 46 can be used to establish a fixed 0,5 probability estimate.

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

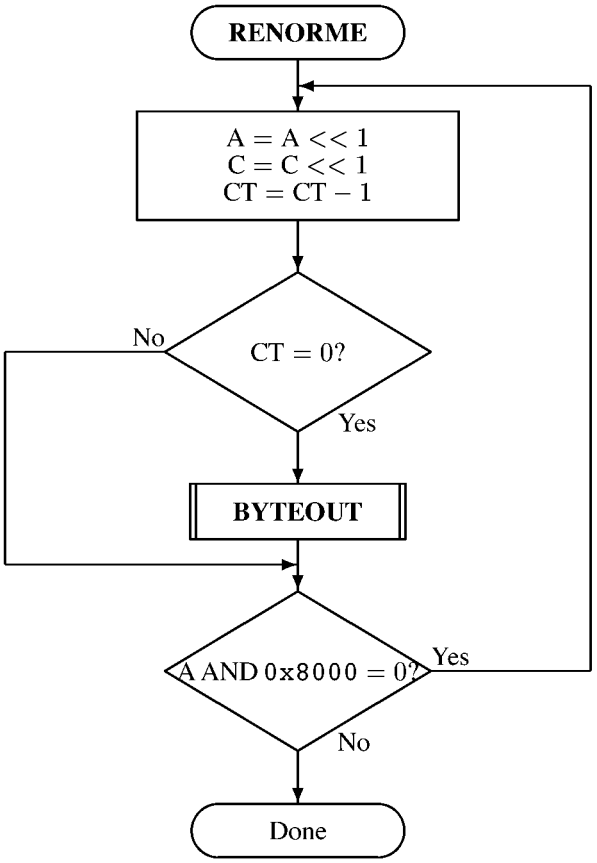


Figure C-8 — Encoder renormalisation procedure

C.2.7 Compressed 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 data. The conventions

used make it impossible for a carry to propagate through more than the byte most recently written to the compressed data buffer.

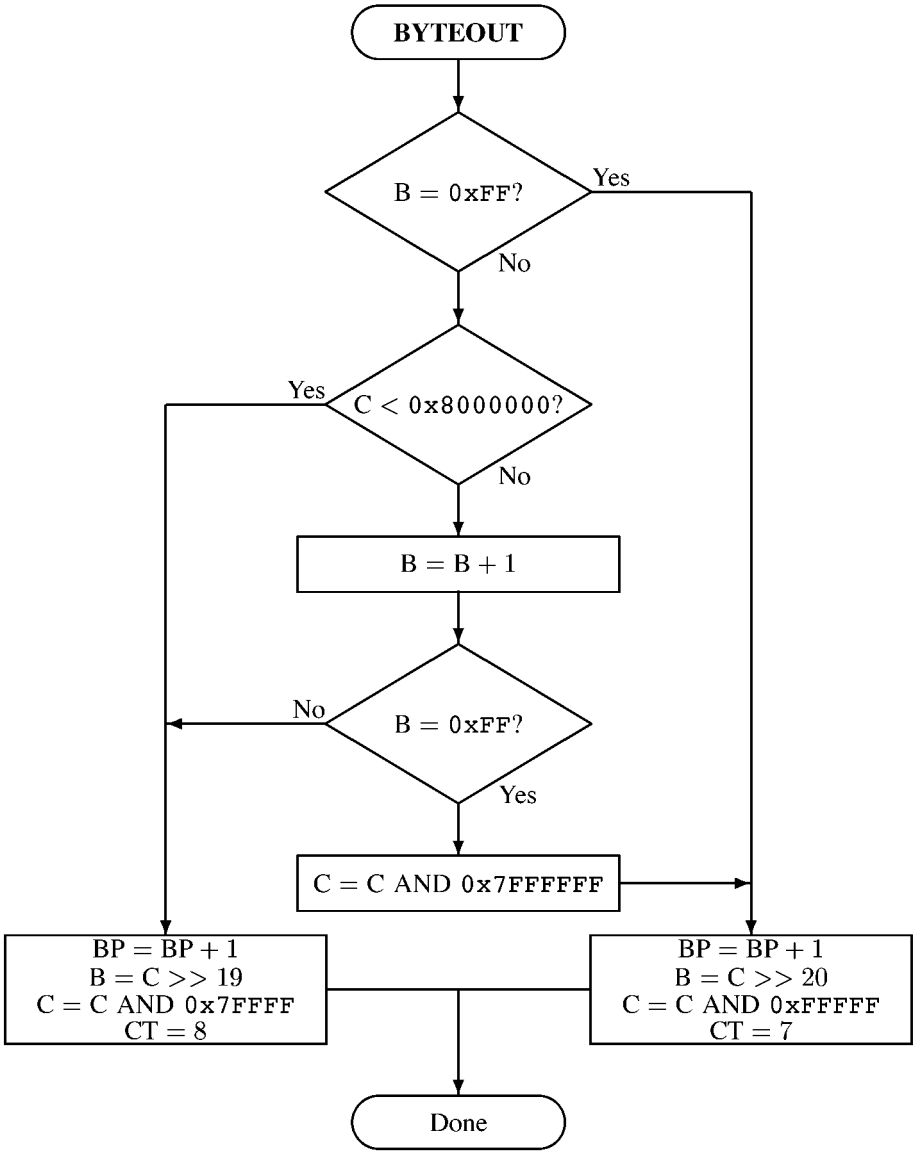


Figure C-9 — BYTEOUT procedure for encoder

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 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 Initialisation of the encoder (INITENC) (informative)

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

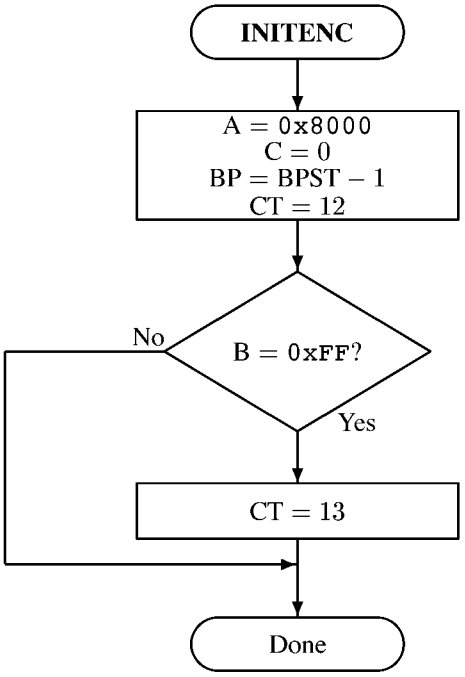


Figure C-10 — Initialisation of the encoder

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

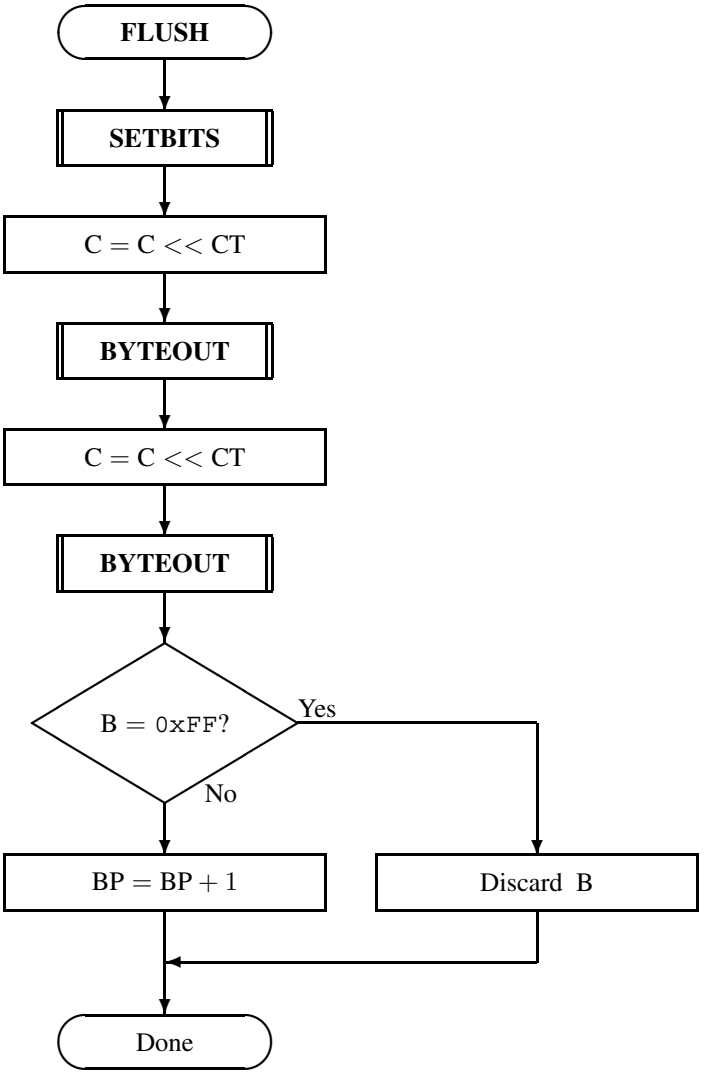


Figure C-11 — FLUSH procedure

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.

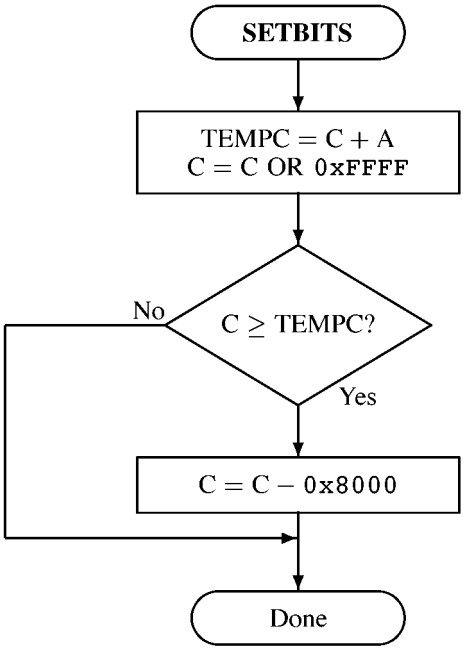


Figure C-12 — Setting the final bits in the C register

The byte in the C-register is then completed by shifting C, and two bytes are then removed. If the byte in buffer, B, 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 provided correct decoding is assured (see Annex D.4.2).

C.3 Arithmetic decoding procedure

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

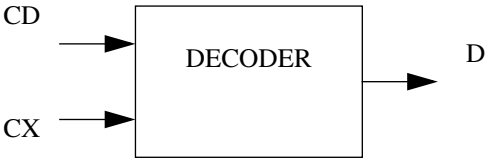


Figure C-13 — Arithmetic decoder inputs and outputs

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

the probability for each context are embedded in DECODE. When all contexts have been read (Finished?), the compressed data has been decompressed.

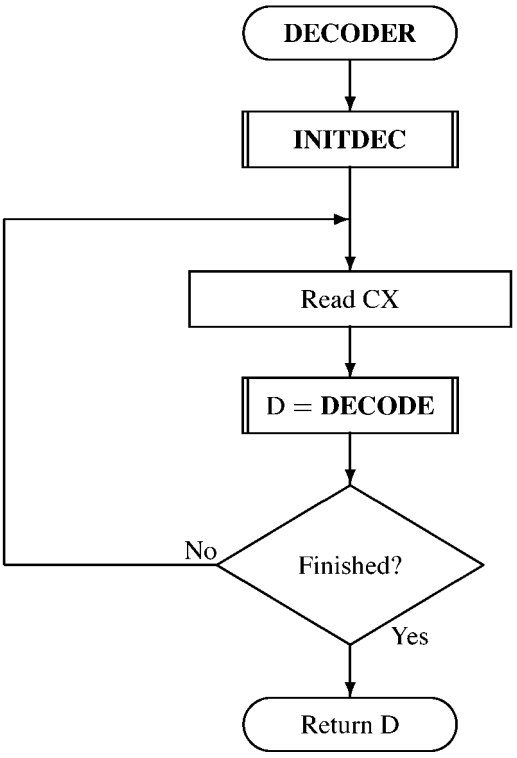


Figure C-14 — Decoder for the MQ-coder

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 section 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 data that the encoder added. The amount left in the compressed data is the offset from the base of the current interval to the sub-interval allocated to all binary decisions not yet decoded. In the first test in the DECODE procedure illustrated in Figure 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.

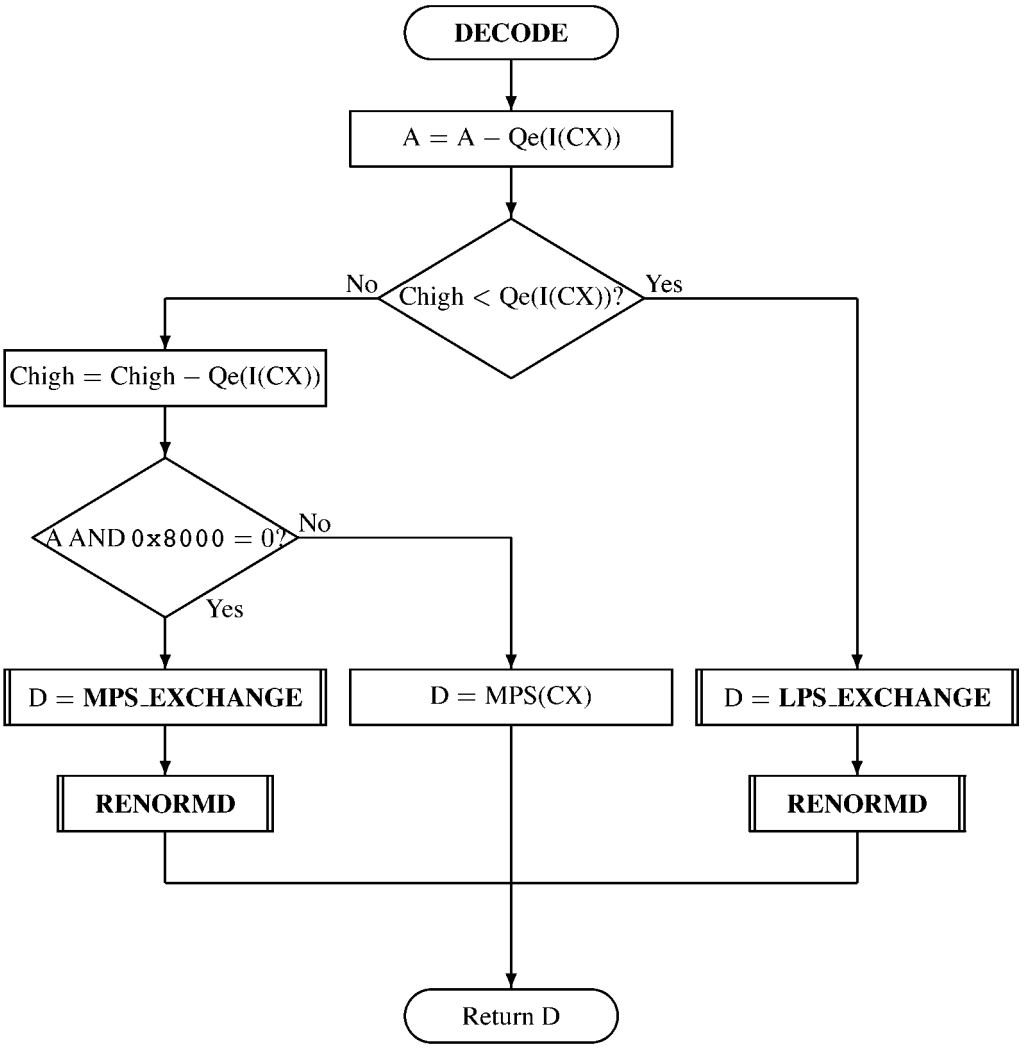


Figure C-15 — Decoding an MPS or an LPS

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. 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. Note that the probability estimation in the decoder needs to be identical to the probability estimation in the encoder.

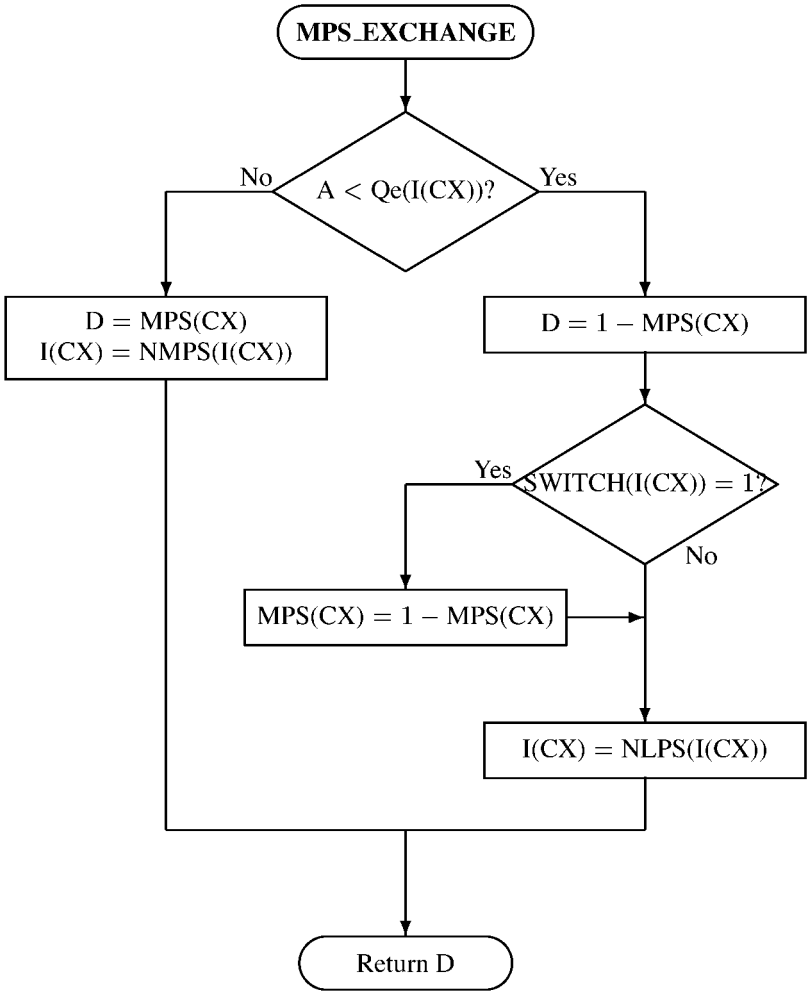


Figure C-16 — Decoder MPS path conditional exchange procedure

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.

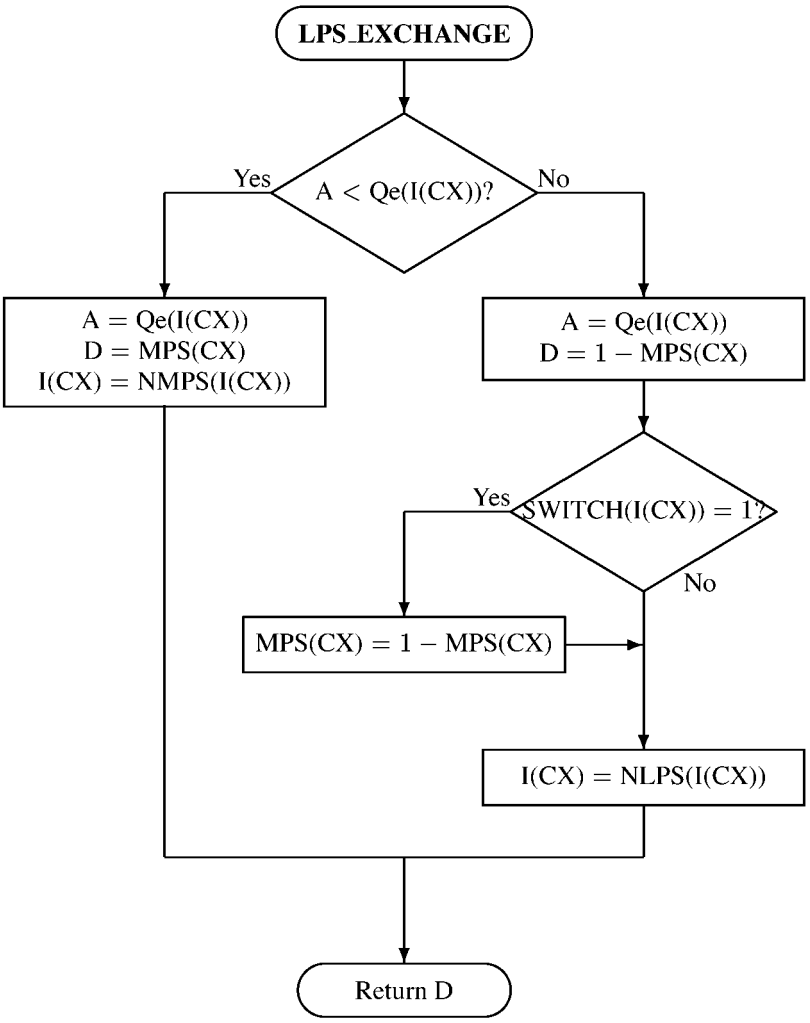


Figure C-17 — Decoder LPS path conditional exchange procedure

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.

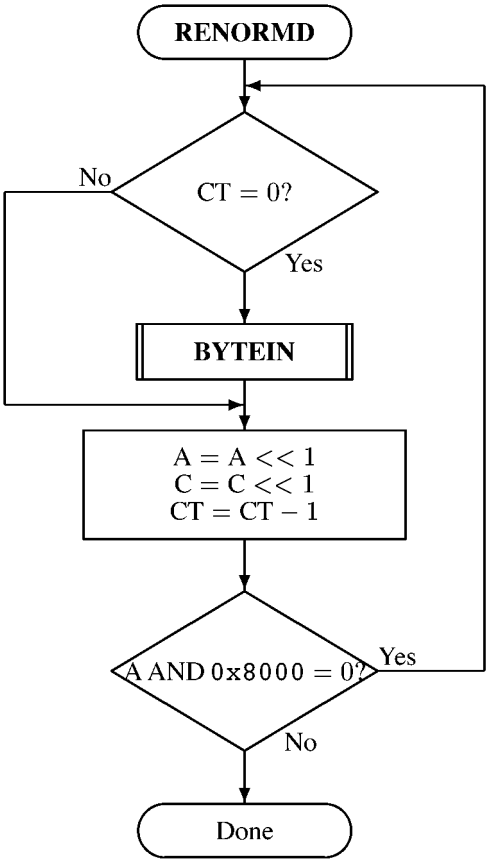


Figure C-18 — Decoder renormalisation procedure

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

C.3.4 Compressed 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 scan or resynchronization interval. The C-register in this procedure is the concatenation of the Chigh and Clow registers.

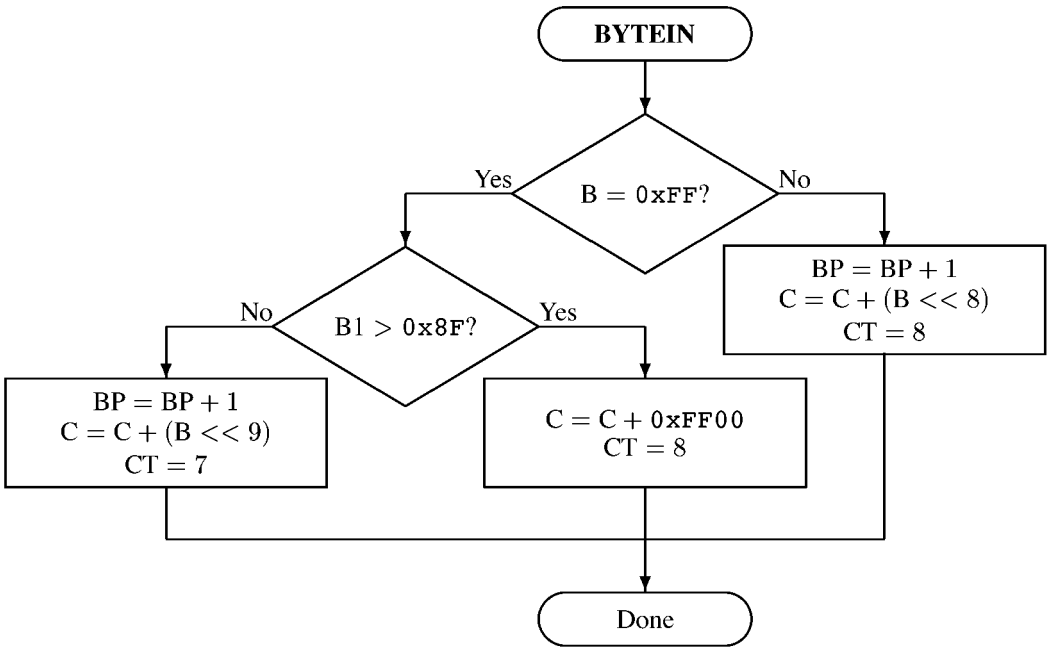


Figure C-19 — BYTEIN procedure for decoder

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

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

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

C.3.5 Initialisation of the decoder (INITDEC)

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

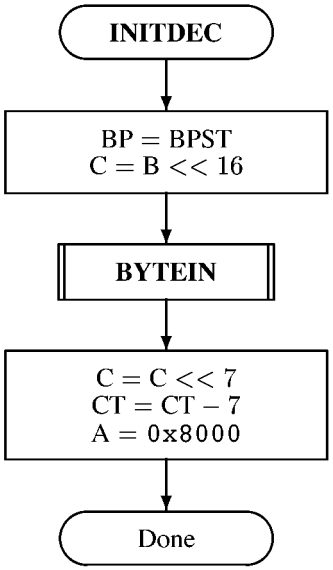


Figure C-20 — Initialisation of the decoder

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

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 setting I(CX) and MPS(CX) equal to zero 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)

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

The coefficients are associated with different sub-bands arising from the transform applied (see Annex F). These coefficients are then arranged into rectangular blocks within each sub-band, called code-blocks. These code-blocks are then coded a bit-plane at a time starting 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 is coded in only one of the three coding passes. The coding passes are 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 Annex C.3). The arithmetic coder is reset according to selected rules.

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 bits of the first column are scanned. Then the first four bits of the second column, until the width of the code-block has been covered. Then the second four bits of the first column are scanned and so on. A similar vertical scan is continued for any leftover rows on the lowest code-blocks in the sub-band. Figure D-1 shows an example of the code-block scan pattern for a code-block.

D.2 Coefficient bits and significance

All quantized transform coefficients, $\bar{q}_b(u, v)$, are signed values even when the original components are unsigned. These coefficients are expressed in a sign-magnitude representation. For a particular sub-band, there is a maximum number of magnitude bits, M_b. The "significance state" changes from insignificant to significant (see the section below) at the bit-plane where the most significant 1 bit is found. For a code-block, the number of bit-planes starting from the most significant bit-plane that are all zero, is signalled in the packet header (see Annex B.9.5). No other coding of those insignificant bit-planes is made.

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 coding of the code-block. The context vector for a given current coefficient is the binary vector consisting of the significance states of its 8 nearest-neighbor coefficients, as shown in Figure D-2. Any nearest neighbor lying outside the

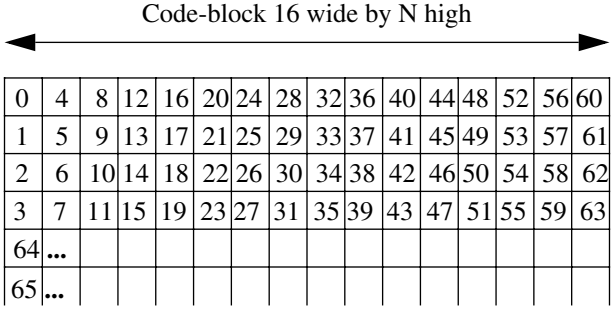


Figure D-1 — Example code-block scan pattern of a code-block

current coefficient's code-block is regarded as insignificant (i.e., it is treated as having a zero significance state) for the purpose of coding a bit in the current coefficient.

In general, a current coefficient can have 256 possible context vectors. These are clustered into a smaller number of contexts according to the rules specified below for context formation. Four different context formation rules are defined, one for each of the four coding operations: significance coding, sign coding, magnitude refinement coding, and cleanup coding. These coding operations are performed in three coding passes over each bit plane: significance and sign coding in a significance propagation 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 number of bit-planes starting from the most significant bit that have no significant coefficients (only insignificant bits) is signalled in the packet headers (see Annex B.9.5). The first bit-plane with a non-zero element has a cleanup pass only. The remaining bit-planes are coded in three coding passes. Each coefficient bit is coded in exactly one of the three coding passes. Which pass a coefficient bit is coded 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 as an X in Figure D-2 where X denotes the current coefficient) are used to create 9 context bins based on how many and which ones are significant. If a coefficient is significant then it is given a 1 value for the creation of the context, otherwise it is given a 0 value. The mapping to the contexts also depends on which sub-band (at a given decomposition level) the code-block is in. Table D-1 shows these contexts.

Table D-1 — Contexts for the significance propagation pass 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]
ΣH	ΣV	ΣD	ΣH	ΣV	ΣD	Σ(H+V)	ΣD	
2	x[b]	x	x	2	x	x	≥3	8
1	≥1	x	≥1	1	x	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

Figure D-2 — Neighbors states used to form the context

Table D-1 — Contexts for the significance propagation pass 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]
0	2	x	2	0	x	1	1	4
0	1	x	1	0	x	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

a. Note that the context labels are numbered only for identification convenience in this specification. The actual identifiers used is a matter of implementation.
b. x = do not care.

The significance propagation pass includes only bits of coefficients that were insignificant (the significance bit has yet to be encountered) and have a non-zero context. All other coefficients are skipped. The context is delivered to the arithmetic decoder (along with the bit stream) and the decoded coefficient bit is returned. 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 of the neighborhood. Computation of the context label can be viewed as a two step process. The first step summarizes the contribution of the vertical and the horizontal neighbors. The second step reduces those contributions to one of 5 context labels.

For the first step, the two vertical neighbors (see Figure 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

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

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

V_0 (or H_0)	V_1 (or H_1)	V (or H) contribution
significant, positive	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 is then logically exclusive ORed with the *XORbit* in Table D-3 to produce the sign bit. The following equation is used:

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

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

Table D-4 — Contexts for the magnitude refinement coding passes

∑H+∑V+∑D	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.

D.3.4 Cleanup pass

All the remaining coefficients are insignificant and had the context value of zero during the significance propagation pass. These are all included in the cleanup 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.

First, the neighbor contexts for the coefficients in this pass are recreated using Table D-1. Note that 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 coded 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.

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 decode with the UNIFORM context are decoded MSB then LSB. That coefficient's sign bit is determined as described in Annex D.3.2. The decoding of any remaining coefficients continues in the manner described in Annex 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. Note that 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 have 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 skip to first coefficient sign skip to second coefficient sign skip to third coefficient sign skip to fourth coefficient sign	MSB LSB 00 01 10 11	 3 2 1 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 coding order for the quantized coefficients of one 4-sample 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 coded. The sign bit is coded after the initial 1 bit and is indicated in the table by the + or - sign. Note that the very first pass in a new block is always a clean-up 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 coded in the significance pass for the next bit-plane. The 1 bit coded for the last coefficient in the second clean-up pass causes the third coefficient to be coded in the next significance pass.

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

Coding Pass	Coefficient Value			
	10	1	3	-7
Clean-up	1+	0	0	0
Significance		0		
Refinement	0			
Clean-up			0	1-
Significance		0	1+	
Refinement	1			1
Clean-up				
Significance		1+		
Refinement	0		1	1
Clean-up				

D.4 Initializing and terminating

When the contexts are initialized, or re-initialized, they are set to the values in the Table D-7. The contexts are either re-initalized at the end of every coding pass, or only at the end of every code-block. The COD or COC marker signals where contexts are reinitialized (see Annex A.6.1 and Annex A.6.2).

Table D-7 — Initial states for all contexts

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

Table D-7 — Initial states for all contexts

Context	Initial index from Table C-2	MPS
All other contexts	0	0

In the normal operation (not selective arithmetic coding bypass), the arithmetic coder shall be terminated either at the end of every coding pass or only at the end of every code-block. 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 Annex A.6.1 and Annex A.6.2).

Table D-8 — Examples of 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 Annex B.9.7.

NOTE — Termination should never create a byte aligned value between 0xFF90 and 0xFFFF. These values are available as in bit steam marker values.

D.4.1 Decoder termination

The decoder anticipates that the given number of codestream bytes will decode a given number of coding passes before the arithmetic coder is terminated. During decoding, bytes are pulled successively from the codestream until all the 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 Annex 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 coder.

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

D.4.2 Arithmetic encoder termination

This termination is required if the predictable termination flag is 1 in the COD or COC markers (see Annex A.6.1 and Annex A.6.2). Otherwise, it is allowed, but not required.

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

The FLUSH procedure performs this task adequately (see Annex 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.

Using the notation of Annex 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 $k = (11 - CT_n) + 1$

2 While $(k > 0)$

— Shift C left by CT and set CT = 0.

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

— Subtract CT from k.

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

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

D.4.3 Length computation (informative)

To include compressed coding pass data into packets the number of bytes to be included must be determined. If the compressed coding pass data is terminated, the algorithm in the previous section 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 (Annex A.6.1 and Annex 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. 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 decoding bypass

This style of coding allows bypassing the arithmetic coder for the significance propagation pass and magnitude refinement coding passes in the fifth significant bit-plane, and the following bit-planes, of the code-block. The first cleanup pass (which is the first bit-plane of a code-block with a non-zero element) and the successive three significance

propagation pass, magnitude refinement, and cleanup coding passes are decoded with the arithmetic coder as before. The fourth cleanup pass shall include an arithmetic coder termination (see Table D-9).

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

The sign bit context is determined as in Annex D.3.2. However, the sign bit is computed with Equation D.2, not Equation D.1.

$$signbit = raw\_value \qquad \text{D.2}$$

where *raw_value* = 1 is a negative sign bit and *raw_value* = 0 is a positive sign bit.

The COD or COC marker signals whether or not this coding style is used (see Annex A.6.1 and Annex A.6.2). Table D-9 shows this progression

Table D-9 — Selective arithmetic coding bypass

#	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

Table D-9 — Selective arithmetic coding bypass

#	Pass type	Coding Operations
final	cleanup	AC, terminate

The length of each terminated segment is signalled in the packet header as described in Annex B.9.7.

D.6.1 Undoing the effects of bit stuffing

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

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

NOTE — Since the decoder appends 0xFF values, as necessary, to the bit stream representing the coding pass (see Annex D.4.1), truncation of the bit stream may be possible.

D.6.2 Predictable termination

This termination is required if the predictable termination flag is 1 in the COD or COC markers (see Annex A.6.1 and Annex A.6.2). Otherwise, it is allowed, but not required. This termination in not optimal.

When all the bits from a coding pass have been assembled by the encoder, if necessary the last byte is packed to a byte boundary with an alternating sequence of 0's and 1's. This sequence should start with a 0 regardless of the number of bits to be padded.

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 samples). 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 Annex A.6.1 and Annex A.6.2).

The bit labelled 14 in Figure D-1 is decoded as usual using the neighbor states as specified in Figure D-2. However, 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 model without the selective arithmetic coding bypass or the vertically causal model.

Table D-10 — Decisions in the context model flow chart

Decision	Question	Description
D0	Is this the first significance bit-plane for the code-block?	Annex D.3
D1	Is the current coefficient significant?	Annex D.3.1
D2	Is the context bin zero? (see Table D-1)	Annex D.3.1
D3	Did the current coefficient just become significant?	Annex D.3.1

Table D-10 — Decisions in the context model flow chart

Decision	Question	Description
D4	Are there more coefficients in the significance propagation?	
D5	Is the coefficient insignificant?	Annex D.3.3
D6	Was the coefficient coded in the last significance propagation?	Annex 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?	Annex D.3.4
D9	Is the coefficient significant?	Annex D.3.4
D10	Are there more coefficients remaining of the four column coefficients?	
D11	Are the four contiguous bits all zero?	Annex D.3.4
D12	Are there more coefficients in the cleanup pass?	

Table D-11 — Coding in the context model flow chart

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

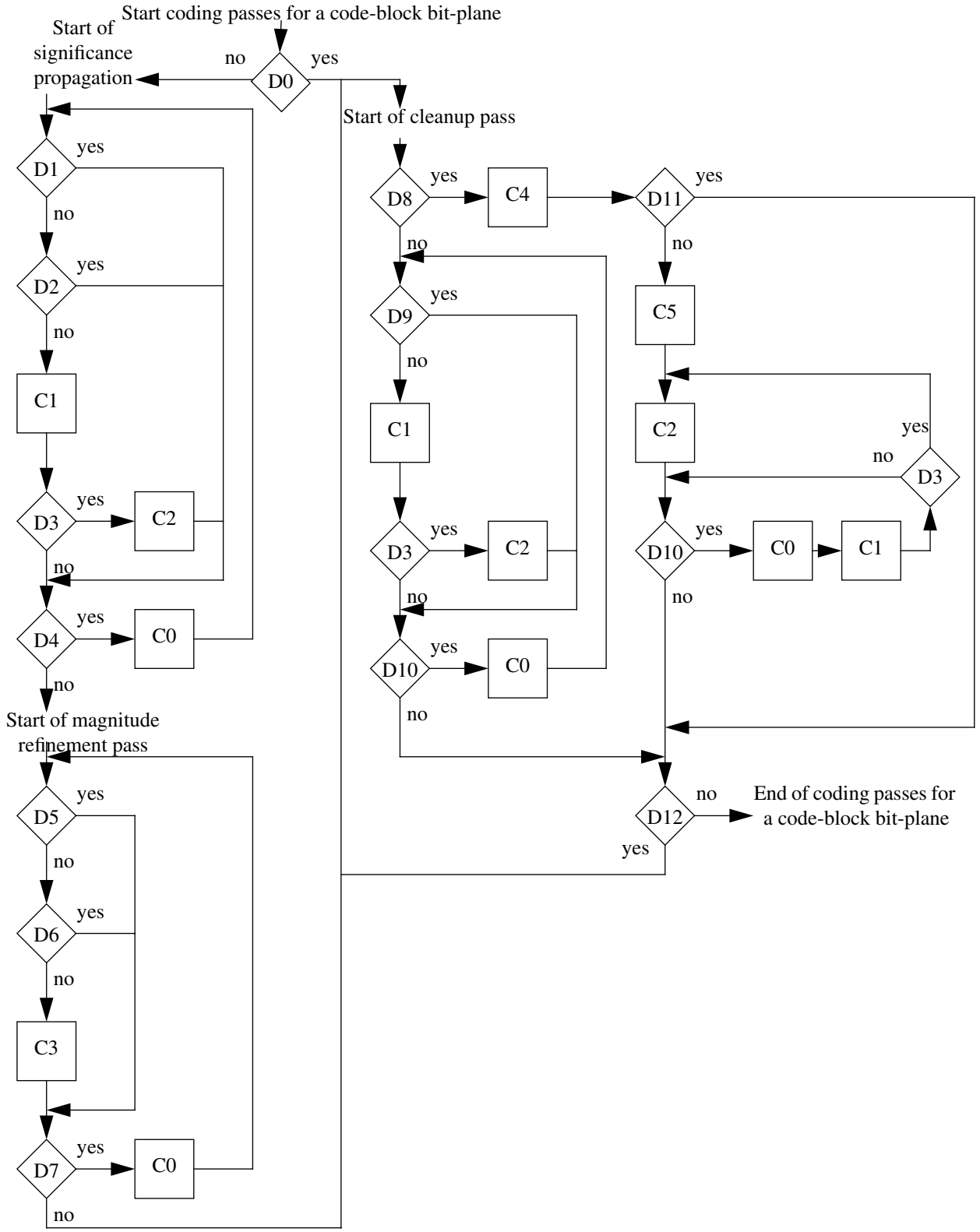


Figure D-3 — Flow chart for all coding passes on a code-block bit-plane

Annex E

Quantization

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

This Annex specifies the forms of quantization and dequantization used for encoding and reconstruction of image tile components. Quantization is the process by which the transform coefficients are reduced in precision. This operation is lossy unless the quantization step is one and the coefficients are integer.

E.1 Scalar coefficient dequantization (normative)

For the 9-7 wavelet filter, the quantization step-sizes for all sub-bands are retrieved from the bit stream using Equation E.1 where ε_b and μ_b are derived from the SPqcd^i parameters defined in the QCD (see Annex A.6.4) or from the SPqcc^i parameters defined in the QCC markers (see Annex A.6.5). The nominal dynamic range R_b is the sum of the number of bits used to represent the original image tile component specified by the SIZ marker (see Annex A.5.1) and the base 2 exponent of the analysis gain of the current sub-band. The analysis gain of a sub-band is recursively defined as the analysis gain of the previous sub-band multiplied by the respective gains of the horizontal and vertical filters used to produce that sub-band. The low-pass filter has a unit gain, while the high-pass filter has a gain of 2. Therefore, the analysis gain of a given sub-band in the wavelet decomposition is 2 to the power of the number of high-pass filtering steps needed to produce that sub-band. Figure E-1 shows the analysis gain of each sub-band for one and two levels of the wavelet transform decomposition and Figure E-2 presents the corresponding nominal dynamic range R_b for each sub-band.

NOTE — The quantized transform coefficient should generally be confined to their nominal dynamic range, but occasional excursions beyond that range might be expected.

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

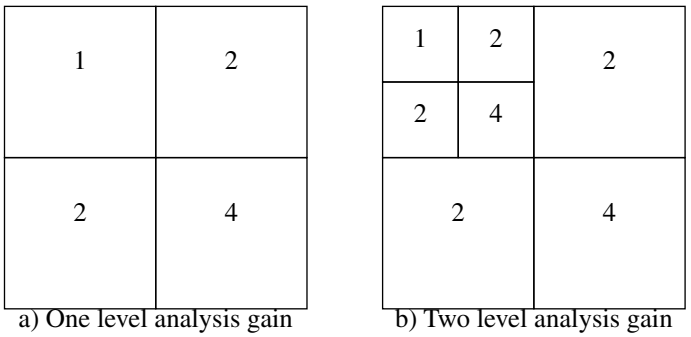
$$\Delta_b = 2^{R_b - \varepsilon_b}\left(1 + \frac{\mu_b}{2^{11}}\right) \qquad \text{E.1}$$

NOTE — The denominator, 2^{11}, in Equation E.1 is determined by the allocation of 11 bits in the codestream for mb, as given in Table A-30.

The exponent/mantissa pairs (ε_b,μ_b) are either explicitly signaled in the bit stream syntax for every sub-band, this is referred to as explicit quantization, or only signaled in the bit stream for the LL band (see Table A-30). In the latter case, known as implicit quantization, all other exponent/mantissa pairs (ε_b,μ_b) are derived implicitly from the single exponent/mantissa pair (ε_o,μ_o) corresponding to the LL band, according to:

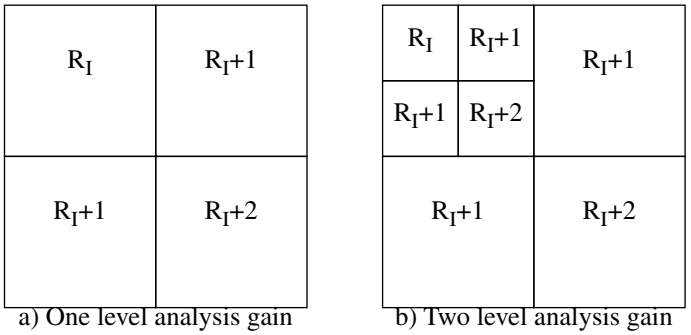
$$(\varepsilon_b,\mu_b) = (\varepsilon_o + nsd_b - nsd_o,\mu_o) \qquad \text{E.2}$$

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



a) One level analysis gain b) Two level analysis gain

Figure E-1 — Analysis gain of each sub-band of the wavelet transform decomposition



a) One level analysis gain b) Two level analysis gain

It is assumed in this example that no region of interest is defined in the image tile component

Figure E-2 — Nominal dynamic range R_b for each sub-band of the wavelet transform decomposition, where R_I is the bit depth of the original image tile-component

The maximum number M_b of encoded bit-planes (see Annex D.1) which can be expected in the code stream for sub-band b is retrieved by using Equation E.3 where the number of guard bits G is specified in the QCD or QCC markers (see Annex A.6.4 and Annex A.6.5).

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

For the reversible 5-3 wavelet transform, the quantization step size is equal to one (no quantization performed). The maximum number M_b of encoded bit-planes is also calculated by Equation E.3, where ε_b is derived from the SPqcd^i parameters defined in the QCD (see Annex A.6.4) or from the SPqcc^i parameters defined in the QCC markers (see Annex A.6.5).

Although the encoder might have encoded all the bit-planes of all samples in sub-band b, due to the embedded nature of the code stream, a decoder may decide to decode only N_b bit-planes for a particular coding-block of the sub-band b. This is equivalent as to the use of a scalar quantizer with step size $2^{M_b - N_b} \cdot \Delta_b$ for all the samples of this coding-block. Due to the nature of the three coding passes (see Annex D.3), the step-size used in practice when truncation of the bit stream occurs may be different for different samples within the same coding-blocks if one bit-plane is not completely decoded. However, these step-sizes are always multiples of the reference step size by some power of two. Each decoded coefficient $\bar{q}_b(u, v)$ of sub-band b is expressed in a sign magnitude representation (see Annex D.3) in which non decoded bits are set to 0. $\bar{q}_b(u, v)$ is used to generate a reconstructed transform coefficient $Rq_b(u, v)$.

For the 9-7 wavelet transform, this reconstructed coefficient is specified in Equation E.4.

$$Rq_b(u,v) = \begin{cases} (\overline{q_b}(u,v) + r2^{M_b - N_b(u,v)}) \cdot \Delta_b & for\ \overline{q_b}(u,v) > 0 \\ (\overline{q_b}(u,v) - r2^{M_b - N_b(u,v)}) \cdot \Delta_b & for\ \overline{q_b}(u,v) < 0 \\ 0 & for\ \overline{q_b}(u,v) = 0 \end{cases} \quad \text{E.4}$$

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

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

In the case of the reversible 5-3 integer wavelet transform, the reconstructed transform coefficient $Rq_b(u,v)$ is recovered differently depending whether the bit stream has been truncated or not. Truncation of the bit stream can be determined from the number of layers signalled in the COD marker in the main or tile header (see Annex A.6.1), and the number of bytes in a code-block is signaled in the packet header (see Annex B.9). If the bit stream is completely decoded (no truncation occurs) then $Rq_b(u,v) = \overline{q_b}(u,v)$ otherwise to reconstruct a transform coefficient $Rq_b(u,v)$, the following formula is used:

$$Rq_b(u,v) = \begin{cases} \left\lfloor (\overline{q_b}(u,v) + r2^{M_b - N_b(u,v)}) \cdot \Delta_b \right\rfloor & for\ \overline{q_b}(u,v) > 0 \\ \left\lfloor (\overline{q_b}(u,v) - r2^{M_b - N_b(u,v)}) \cdot \Delta_b \right\rfloor & for\ \overline{q_b}(u,v) < 0 \\ 0 & for\ \overline{q_b}(u,v) = 0 \end{cases} \quad \text{E.5}$$

E.2 Scalar coefficient quantization (informative)

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

$$q_b(u,v) = sign(a_b(u,v)) \cdot \left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor \quad \text{E.6}$$

where the quantization step size Δ_b is represented using Equation E.1

In order to prevent possible overflow or excursion beyond the nominal range of the integer representation of $|q_b(u,v)|$ arising, for example during floating point calculations, the number of bits for the integer representation of $q_b(u,v)$ used at the encoder side is defined by Equation E.3. The number G of guard bits, has to be specified in the QCD or QCC marker (see Annex A.6.4 and Annex A.6.5). If a ROI is defined, then the number of magnitude bit is modified accordingly (see Annex H).

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

For reversible compression, the quantization step size is required to be 1. This implies that $\mu_b = 0$ and $R_b = \varepsilon_b$. In this case, only the exponent ε_b has to be recorded in the bit stream in the QCD or QCC markers (see Annex A.6.4 and Annex A.6.5).

NOTE — When the RCT is used the nominal dynamic range has to be modified according to Annex G.

For irreversible compression, no particular selection of the quantization step size is required in this Specification and different applications may specify the quantization step sizes according to specific image tile component characteristics. One effective way of selecting the quantizer step size for one sub-band b is to normalize a default step size Δ_d with respect to the vertical and horizontal synthesis filters which are used in that specific sub-band [22]. The relationship between errors in one transformed coefficient induced by quantization and the corresponding errors in the samples value in the image tile component is expressed by the energy weight γ_b, i.e. the amount of squared errors introduced by a unit error in the transformed coefficient. The energy weight of a given sub-band b is the product of its row weight and column weight. The column (row) weight is a function of the synthesis filter applied in the column (row) direction during

the Inverse Transform (see Annex F). For example, a transformed coefficient belonging to sub-band $b = 3LH$ (see Annex F for the definition of 3LH) undergoes three low-pass filtering in the row direction. In the column direction, the appropriate filtering is high-pass followed by two low-pass. Let l_p and h_p be the impulse response of the low and high pass synthesis 1D filters (see Table E.1). To calculate the column weight, h_p is up-sampled (one zero is inserted between every coefficient of the filter) and convolved with l_p. The result is then up-sampled and convolved with l_p. If more than three synthesis filters are applied in the column direction, the previous calculation is repeated until all filters needed to perform the inverse wavelet transform have been applied. The column weight is then the sum of the square of all samples in the final convolution result. The row weight is computed in the same way. A typical choice for the quantization step size for sub-band $b = 3LH$ is then:

$$\Delta_b = \frac{\Delta_d \cdot 2^{R_b}}{\sqrt{\gamma_b}} \qquad \text{E.7}$$

A typical value for Δ_d is 2^{1-R_I} where R_I is the bit depth of the original tile image component.

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

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

Annex F

Discrete wavelet transformation of tile components

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

This Recommendation | International Standard 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 Introduction and overview

F.1.1 Tile component parameters

Consider the tile component defined by the coordinates tcx_0, tcx_1, tcy_0 and tcy_1 given in Equation B.10, in Annex B. Then the coordinates (x, y) of the tile component (with sample values $I(x, y)$) lie in the range defined by:

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

F.1.2 Discrete Wavelet Transformations (informative)

F.1.2.1 Low-pass and high-pass filtering

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

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

F.1.2.2 Levels of decomposition

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. The different number of levels of spatial resolutions N_L is called the number of decomposition levels.

F.1.2.3 Discrete wavelet filters (informative)

This Recommendation | International Standard uses 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.2 The inverse discrete wavelet transformation (normative)

F.2.1 The IDWT procedure

The inverse discrete wavelet transformation (IDWT) inverse transforms a set of sub-bands with coefficients $a_b(u_b, v_b)$ into DC-level shifted tile component samples $I(x, y)$ (IDWT procedure), which depend on the parameter N_L,

representing a number of iterations, known as 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 Annex A.6.1 and Annex A.6.2).

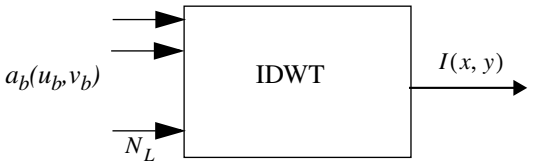


Figure F-1 — Inputs and outputs of the IDWT procedure

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

The following ordering of sub-bands is used:

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

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

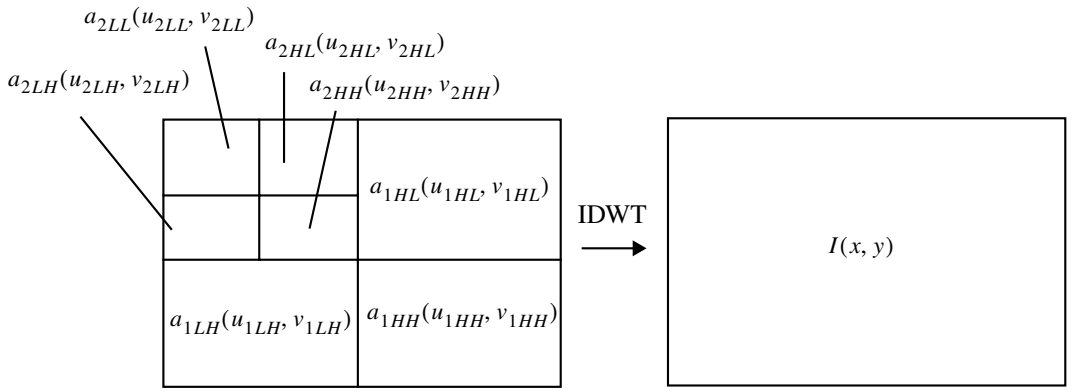


Figure F-2 — The IDWT (N_L=2)

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

As defined in Annex B, the coefficient values $a_{levLL}(u, v)$ lie in the range defined by:

$$tbx_0 \le u < tbx_1 \text{ and } tby_0 \le v < tby_1, \qquad \text{F.2}$$

which are defined in Annex B.

Figure F-3 describes the IDWT procedure.

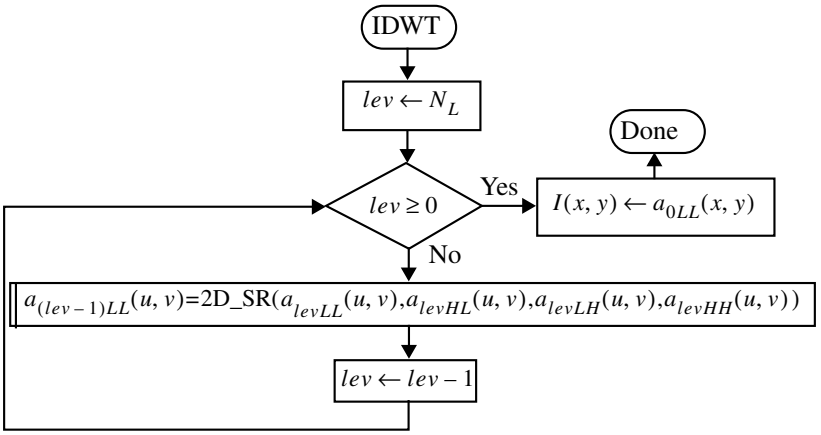


Figure F-3 — The IDWT Procedure

F.2.2 The 2D_SR procedure

The 2D_SR procedure performs a recomposition of four groups of sub-band coefficients $a_{(lev+1)LL}(u, v)$, $a_{(lev+1)HL}(u, v)$, $a_{(lev+1)LH}(u, v)$ and $a_{(lev+1)HH}(u, v)$ into a two-dimensional array of samples $a_{levLL}(u, v)$ (see Figure F-4).The total number of samples of the recomposed $levLL$ sub-band is equal to the sum of the total number of samples of the four sub-bands input to the 2D_SR procedure (see Figure F-5).

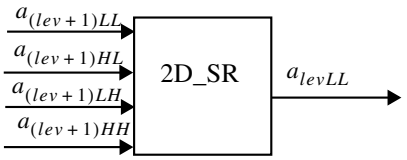


Figure F-4 — Inputs and outputs of the 2D_SR procedure

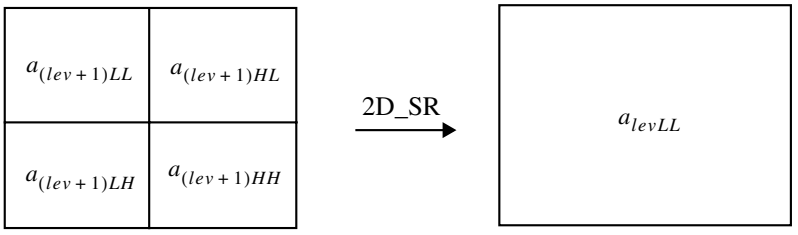


Figure F-5 — One-level recomposition from four sub-bands (2D_SR procedure)

First, the four sub-bands are interleaved to form an array $a(u, v)$ using the 2D_INTERLEAVE procedure. Then the 2D_SR procedure first applies the HOR_SR procedure to all rows of $a(u, v)$. It finally applies the VER_SR procedure to all columns of $a(u, v)$.

Figure F-6 describes the 2D_SR procedure.

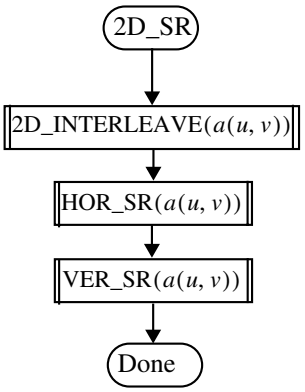


Figure F-6 — The 2D_SR procedure

F.2.3 The 2D_INTERLEAVE procedure

As illustrated in Figure F-7, the 2D_INTERLEAVE procedure interleaves the coefficients of four sub-bands to form $a(u, v)$.

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

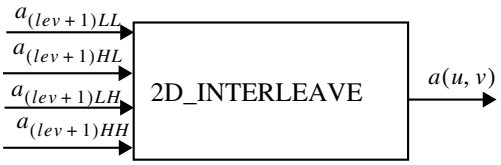


Figure F-7 — Parameters of 2D_INTERLEAVE procedure

F.2.4 The HOR_SR procedure

The HOR_SR procedure performs a horizontal sub-band recomposition of a two-dimensional array of samples. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical coordinates (u_0, u_1) and (v_0, v_1) of its first and last samples (see Figure F-9) and produces as output a horizontally inverse filtered version of the input array, row by row.

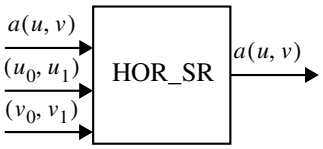


Figure F-9 — Inputs and outputs of the HOR_SR procedure

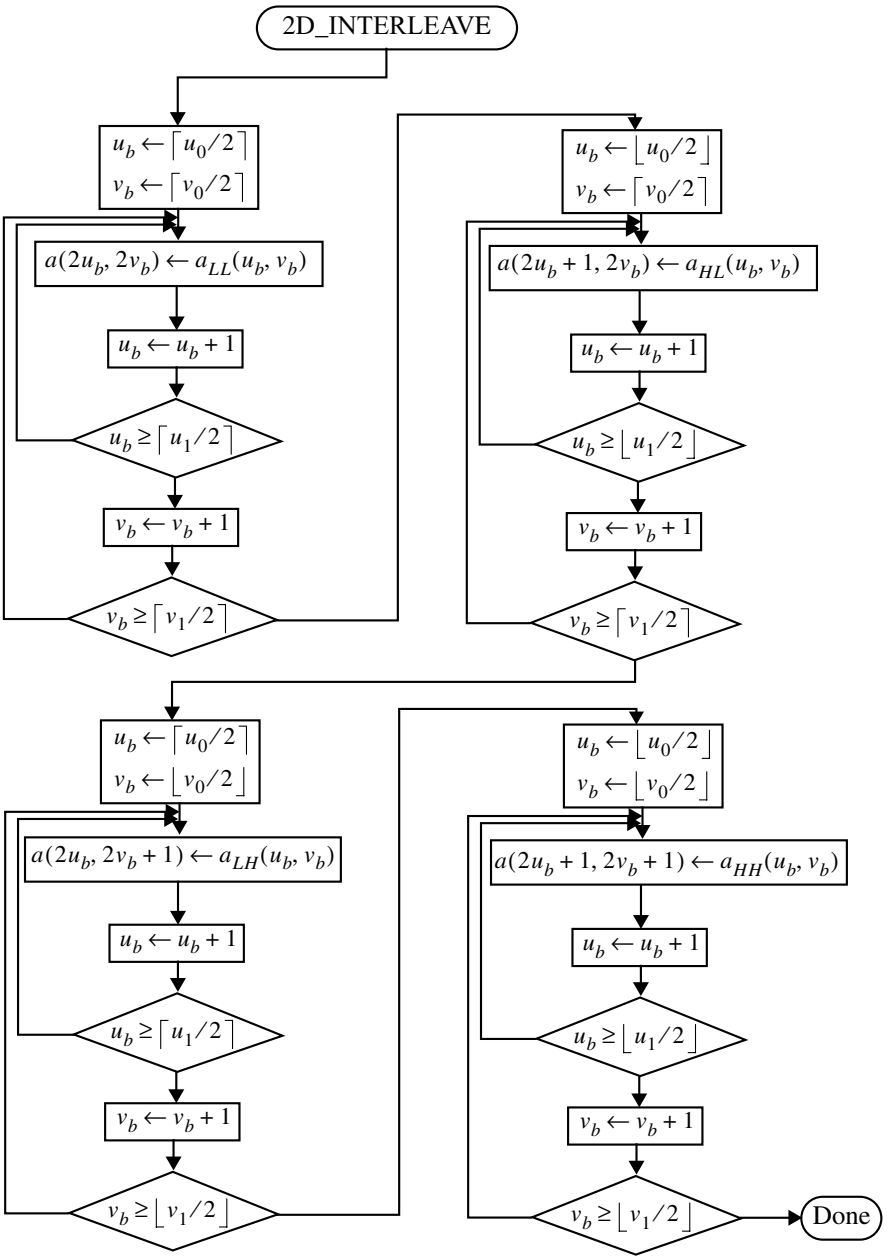


Figure F-8 — The 2D_INTERLEAVE procedure

As illustrated in Figure F-10, the HOR_SR procedure applies the one-dimensional sub-band recomposition (1D_SR procedure) to each row of the input array $a(u, v)$. The result of the application of the 1D_SR procedure on each row is stored back in each row.

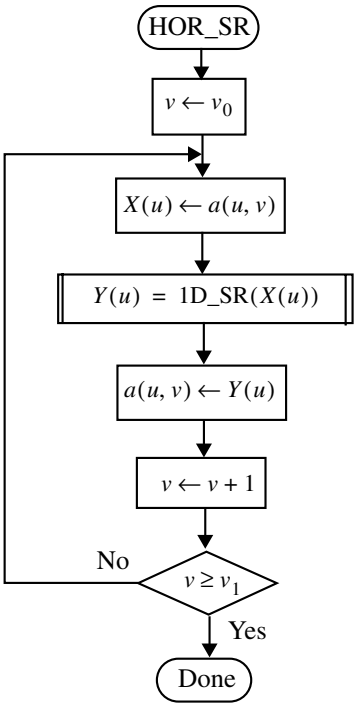


Figure F-10 — The HOR_SR procedure

F.2.5 The VER_SR procedure

The VER_SR procedure performs a vertical sub-band recomposition of a two-dimensional array of samples. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical coordinates (u_0, u_1) and (v_0, v_1) of its first and last samples (see Figure F-11) and produces as output a vertically inverse filtered version of the input array, column by column.

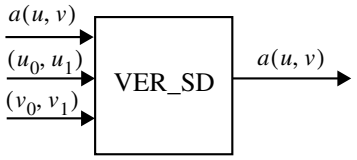


Figure F-11 — Inputs and outputs of the VER_SR procedure

As illustrated in Figure F-12, the VER_SR procedure applies the one-dimensional sub-band recomposition (1D_SR procedure) to each column of the input array $a(u, v)$. The result of the application of the 1D_SR procedure on each column is stored back in each column.

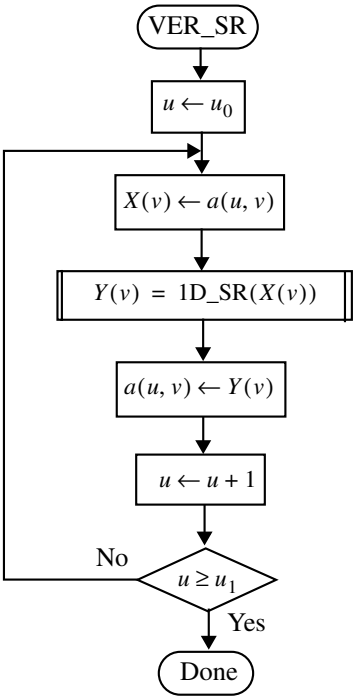


Figure F-12 — The VER_SR procedure

F.2.6 The 1D_SR procedure

As illustrated in Figure F-13, the 1D_SR procedure takes as input a one-dimensional array *Y*, the index i_0 of the first sample in array *X*, the index i_1 of the sample following the last sample in array *Y*. It produces as output an array *X*, with the same indices (i_0, i_1).

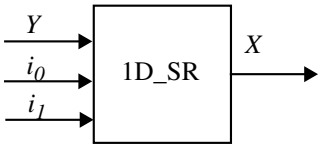


Figure F-13 — Parameters of the 1D_SR procedure

For signals of length one (i.e. $i_0 = i_1 - 1$), the 1D_SR procedure is the identity, i.e. $X(i_0) = Y(i_0)$.

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_IFILTR procedure to inverse filter the extended signal Y_{ext} and produce the desired filtered signal X.

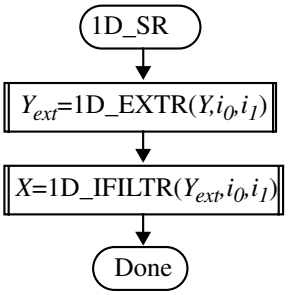


Figure F-14 — The 1D_SR procedure

F.2.7 The 1D_EXTR procedure

As illustrated in Figure F-15 and Figure F-16, the 1D_EXTR procedure extends signal X by i_{left} samples to the left and i_{right} samples to the right. The extension of the signal is needed to enable filtering at both boundaries of the signal.

The first sample of signal X is sample i_0, and the last sample of signal X is sample i_1-1. This extension procedure is known as “periodic symmetric extension”. Symmetric extension consists in extending the signal with the signal samples obtained by a reflection of the signal centered on the first sample (sample i_0) for extension to the left, and in extending the signal with the signal samples obtained by a reflection of the signal centered on the last sample (sample i_1-1) for extension to the right. Periodic symmetric extension applies to the case where the number of samples by which to extend the signal on any one side exceeds the signal length: this case may happen at lower levels of decomposition. The procedure described in Figure F-15 is one among possibly others which implements periodic symmetric extension.

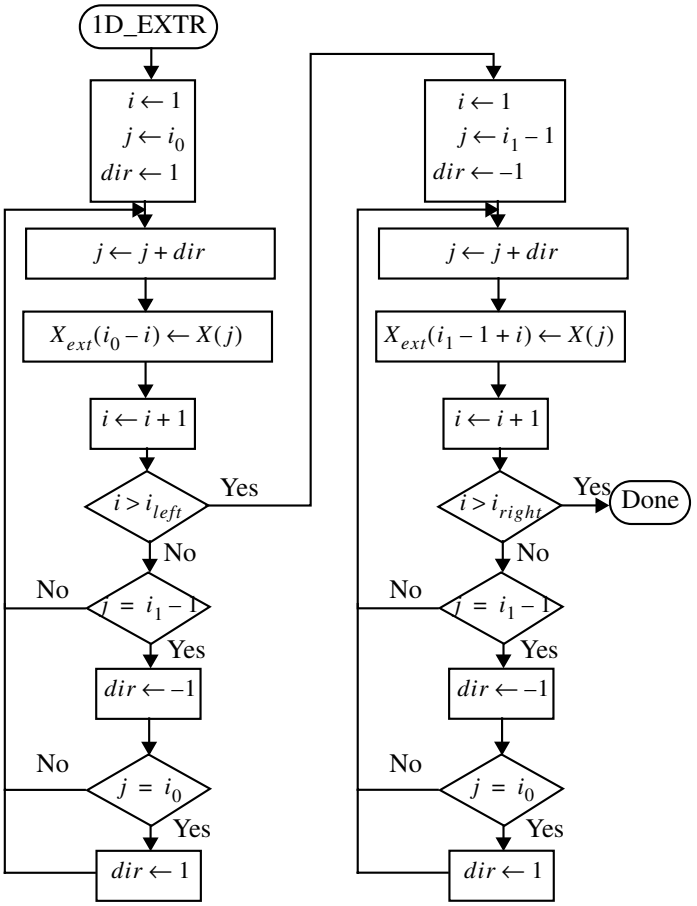


Figure F-15 — 1D_EXTR procedure implementing periodic symmetric extension

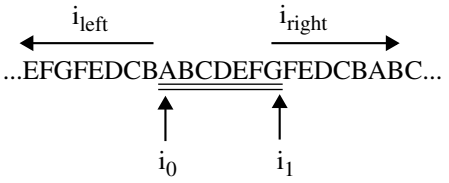


Figure F-16 — Periodic symmetric extension of signal

The minimum but sufficiently large values of the extension parameters i_{left} and i_{right} for the reversible transformation (5/3) and the irreversible transformation (9/7) are given in Table F-1 and Table F-2

Table F-1 — Extension to the left

i_0	$i_{left}(5/3)$	$i_{left}(9/7)$
even	2	4
odd	1	3

and Table F-1

Table F-2 — Extension to the right

i_1	$i_{right}(5/3)$	$i_{right}(9/7)$
odd	2	4
even	1	3

F.2.8 The 1D_IFILTR procedure

One irreversible inverse filtering procedure $1D\_IFILTR_I$ and one reversible filtering procedure $1D\_IFILTR_R$ are described.

As illustrated in Figure F-17, 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.

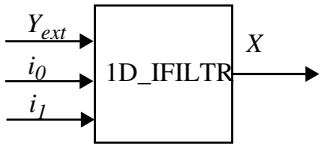


Figure F-17 — Parameters of the 1D_IFILTR procedure

Both the irreversible and reversible inverse transformations are described using lifting-based inverse filtering [14], which is a very efficient implementation of the inverse DWT. Lifting-based filtering consists of a sequence of very simple filtering operations for which alternately, odd coefficient values of the signal are updated with a weighted sum of even coefficient values, and even coefficient values are updated with a weighted sum of odd coefficient values.

F.2.8.1 Reversible 1D inverse filtering

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

The even sample values of output signal X are computed first from the even coefficient values of extended signal Y_{ext} and the odd coefficient values of signal Y_{ext} for all values of n such that $i_0 - 1 \le 2n < i_1 - 1$:

$$X(2n) = Y_{ext}(2n) - \left\lfloor \frac{Y_{ext}(2n-1) + Y_{ext}(2n+1) + 2}{4} \right\rfloor \qquad \text{F.3}$$

Then the odd sample values of output signal X are computed from the odd coefficient values of extended signal Y_{ext} and the even sample values of signal X for all values of n such that $i_0 \le 2n + 1 < i_1$:

$$X(2n+1) = Y_{ext}(2n+1) + \left\lfloor \frac{X(2n) + X(2n+2)}{2} \right\rfloor. \qquad \text{F.4}$$

F.2.8.2 Irreversible 1D inverse filtering

The irreversible inverse transformation described in this section is the lifting-based DWT implementation of filtering by the Daubechies 9/7 filter [6].

Equation F.5 describes the 2 “scaling” steps (1 and 2) and the 4 “lifting” steps (3 through 6) and of the 1D filtering performed on the extended signal $Y_{ext}(n)$ to produce the i_1 samples of signal X.

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

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

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

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

$$\begin{cases} \alpha = -1{,}586\ 134\ 342 \\ \beta = -0{,}052\ 980\ 118 \\ \gamma = 0{,}882\ 911\ 075 \\ \delta = 0{,}443\ 506\ 852 \end{cases} \quad \text{F.6}$$

and the scaling factor *K* is equal to: $K = 1,230\ 174\ 105$.

F.3 Forward Transformation (informative)

F.3.1 The FDWT procedure

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), which depend on the parameter N_L , representing a number of iterations, known as the number of decomposition levels (see Figure F-18). The number of decomposition levels N_L is signalled in the COD or COC markers (see Annex A.6.1 and Annex A.6.2).

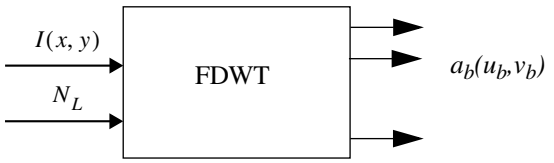


Figure F-18 — Inputs and outputs of the FDWT procedure

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

The following ordering of sub-bands is used:

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

As illustrated in Figure F-19, all the sub-bands in the case where N_L=2 can be represented in the following way:

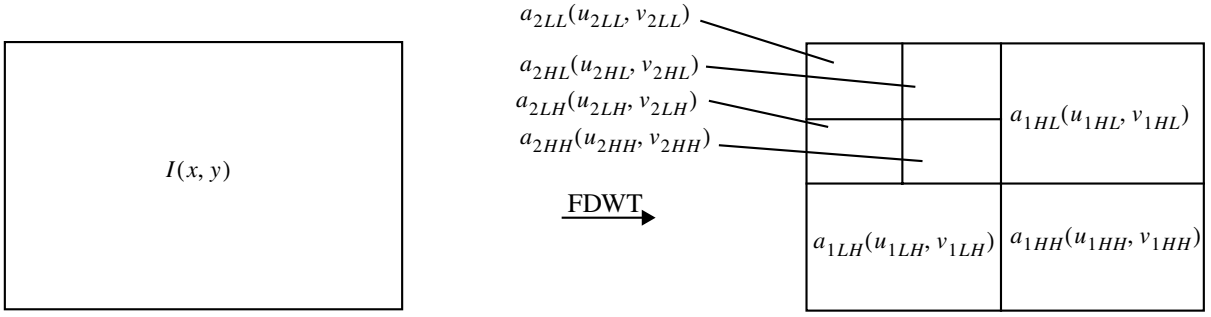


Figure F-19 — The FDWT (N_L=2)

The FDWT procedure starts with the initialization of the variable lev (the current level of decomposition) to zero, and and setting the sub-band $a_{0LL}(u, v)$ to the input array $I(u, v)$. The 2D_SD procedure is performed at every level lev, where the level lev increases at each iteration, and until N_L iterations are performed. The 2D_SD procedure is iterated over the LL sub-band produced at each iteration.

As defined in Annex B, the coordinates of the sub-band $a_{levLL}(u, v)$ lie in the range defined by:

$$tbx_0 \le u < tbx_1 \quad \text{and} \quad tby_0 \le v < tby_1 \ . \qquad \text{F.7}$$

Figure F-20 describes the FDWT procedure.

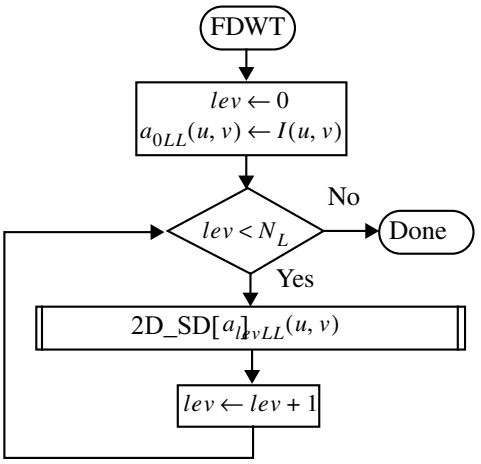


Figure F-20 — The FDWT Procedure

F.3.2 The 2D_SD procedure

The 2D_SD procedure performs a decomposition of a two-dimensional array of samples $a_{levLL}(u, v)$ into four groups of sub-band coefficients $a_{(lev+1)LL}(u, v)$, $a_{(lev+1)HL}(u, v)$, $a_{(lev+1)LH}(u, v)$ and $a_{(lev+1)HH}(u, v)$. The four sub-bands are filtered and downsampled version of the original array of samples.

The total number of samples of the $levLL$ sub-band is equal to the sum of the total number of samples of the four sub-bands resulting from the 2D_SD procedure.

Figure F-21 describes the input and output parameters of the 2D_SD procedure.

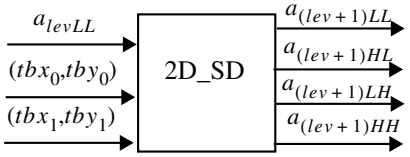


Figure F-21 — Inputs and outputs of the 2D_SD procedure

Figure F-22 illustrates the sub-band decomposition performed by the 2D_SD procedure.

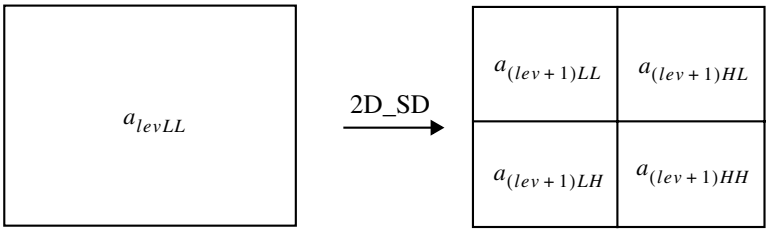


Figure F-22 — One-level decomposition into four sub-bands (2D_SD procedure)

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-23 describes the 2D_SD procedure.

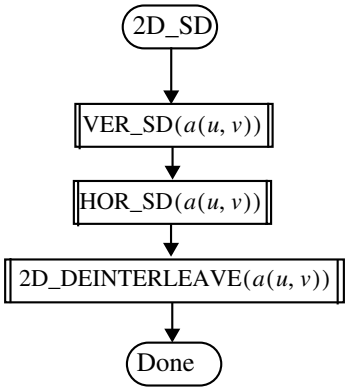


Figure F-23 — The 2D_SD procedure

F.3.3 The VER_SD procedure

The VER_SD procedure performs a vertical sub-band decomposition of a two-dimensional array of samples. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical coordinates (u_0, u_1) and (v_0, v_1) of its first and last samples (see Figure F-24) and produces as output a vertically filtered version on the input array, column by column.

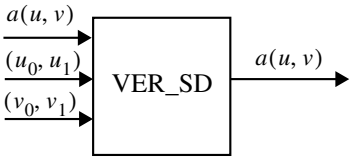


Figure F-24 — Inputs and outputs of the VER_SD procedure

As illustrated in Figure F-25, the VER_SD procedure applies the one-dimensional sub-band decomposition (1D_SD procedure) to each column of the input array $a(u, v)$. The result of the application of the 1D_SD procedure on each column is stored back in each column.

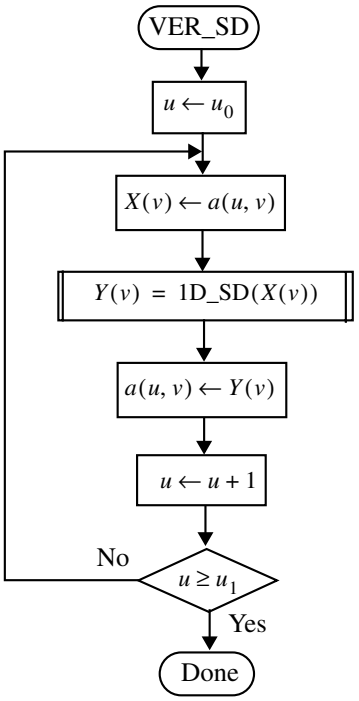


Figure F-25 — The VER_SD procedure

F.3.4 The HOR_SD procedure

The HOR_SD procedure performs a horizontal sub-band decomposition of a two-dimensional array of samples. It takes as input a two-dimensional array $a(u, v)$, the horizontal and vertical coordinates (u_0, u_1) and (v_0, v_1) of its first and last samples (see Figure F-26) and produces as output a horizontally filtered version on the input array, row by row.

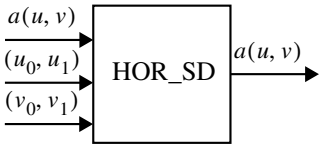


Figure F-26 — Inputs and outputs of the HOR_SD procedure

As illustrated in Figure F-27, the HOR_SD procedure applies the one-dimensional sub-band decomposition (1D_SD procedure) to each row of the input array $a(u, v)$. The result of the application of the 1D_SD procedure on each row is stored back in each row.

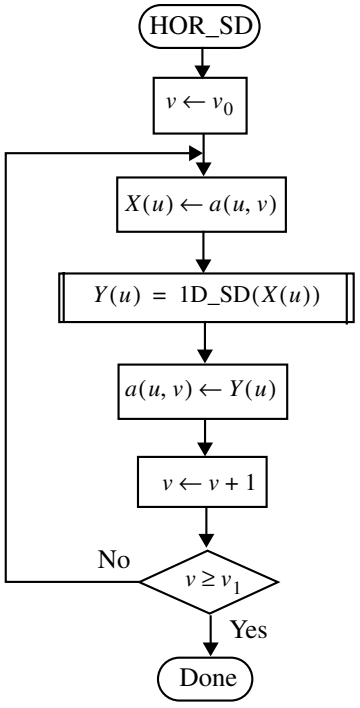


Figure F-27 — The HOR_SD procedure

F.3.5 The 2D_DEINTERLEAVE procedure

As illustrated in Figure F-28, 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 sample 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-28 — Parameters of 2D_DEINTERLEAVE procedure

The formation of the sub-bands is simply a deinterleaving of the coefficients of $a(u, v)$.

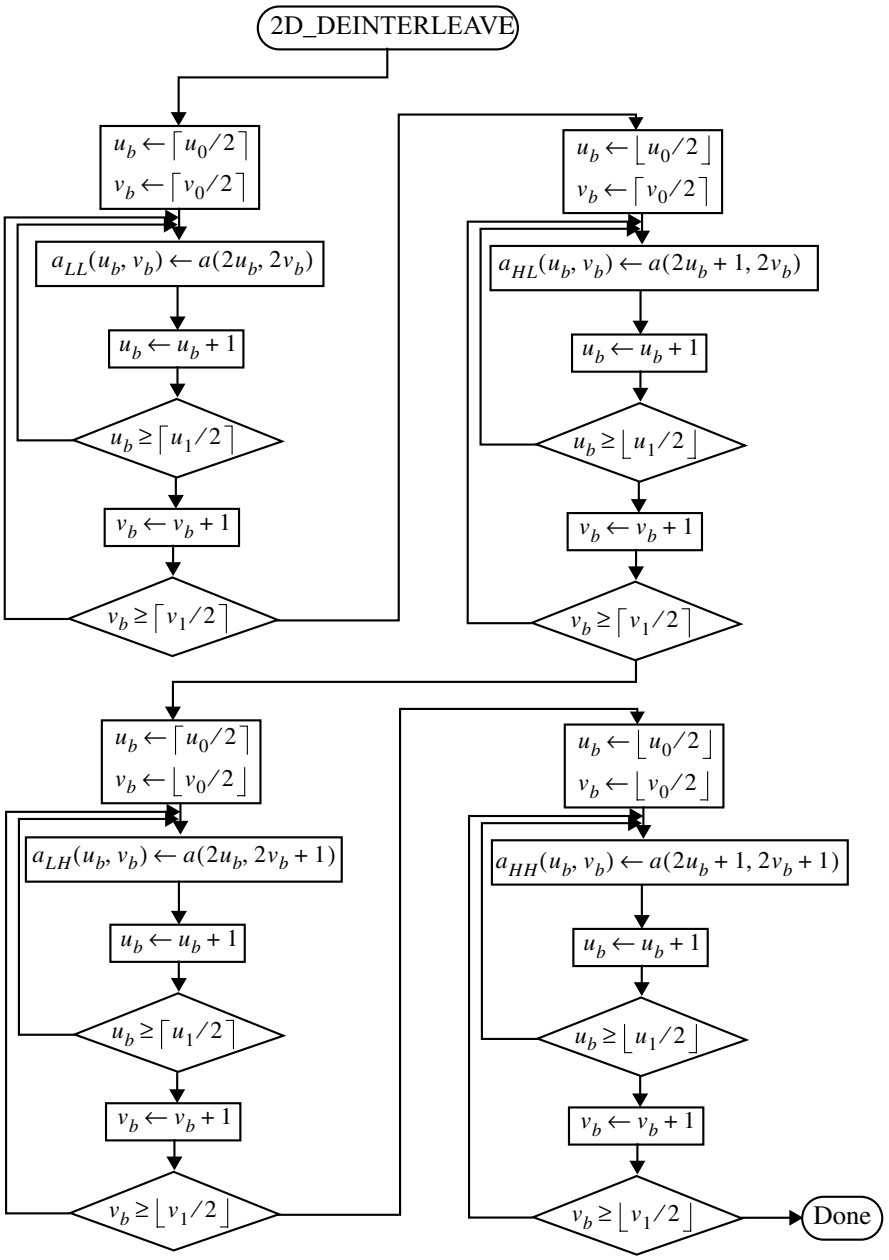


Figure F-29 — The 2D_DEINTERLEAVE procedure

F.3.6 The 1D_SD procedure

As illustrated in Figure F-30, the 1D_SD procedure takes as input a one-dimensional array X, the index i_0 of the first sample in array X, the index i_1 of the sample following the last sample in array X. It produces as output an array Y, with the same indices (i_0, i_1).

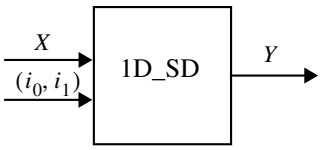


Figure F-30 — Parameters of the 1D_SD procedure

For signals of length one (i.e. $i_0 = i_1 - 1$), the 1D_SD procedure is the identity, i.e. $Y(i_0) = X(i_0)$.

For signals of length greater than or equal to two (i.e. $i_0 < i_1 - 1$), as illustrated in Figure F-31, 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.

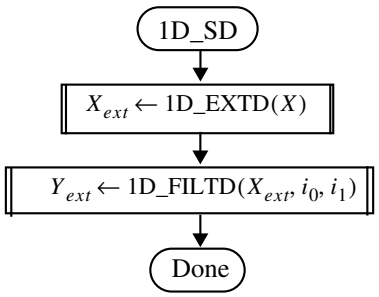


Figure F-31 — The 1D_SD procedure

F.3.7 The 1D_EXTD procedure

The 1D_EXTD procedure is identical to the 1D_EXTR procedure.

F.3.8 The 1D_FILTD procedure

This Recommendation | International Standard uses exclusively one irreversible or one reversible transformation of image tile components. The transformation is reversible if the 1D_FILTD procedure is reversible. The transformation is irreversible if the 1D_FILTD procedure is irreversible. One irreversible procedure $1D\_FILTD_I$ and one reversible filtering procedure $1D\_FILTD_R$ is described.

As illustrated in Figure F-32, both procedures take as input an extended 1D signal X_{ext}, the index of the first sample i_0, and the index of the sample i_1 immediately following the last sample (i_1-1). They both produce the output signal Y. The even coefficients of the Y signal are a low-pass downsampled version of the extended signal X_{ext}, while the odd coefficients of the signal Y are a high-pass downsampled version of the extended signal X_{ext}.

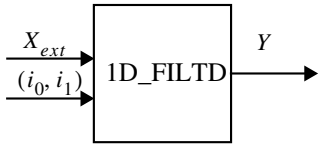


Figure F-32 — Parameters of the 1D_FILTD procedure

Both the irreversible and reversible transformations are described using lifting-based filtering [14], which is a very efficient implementation of the DWT. Lifting-based filtering consists of a sequence of very simple filtering operations for

which alternately, odd sample values of the signal are modified with a weighted sum of even sample values, and even sample values are modified with a weighted sum of odd sample values.

F.3.8.1 Reversible 1D filtering

The reversible transformation described in this section is the reversible lifting-based implementation of filtering by the 5/3 filter [13].

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

The odd coefficients of output signal Y are computed first for all values of n such that $i_0-1 \le 2n+1 < i_1+1$:

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

Then the even coefficients of output signal Y are computed from the even values of extended signal X_{ext} and the odd coefficients of signal Y for all values of n such that $i_0 \le 2n < i_1$

$$Y(2n) = X_{ext}(2n) + \left\lfloor \frac{Y(2n-1) + Y(2n+1) + 2}{4} \right\rfloor \qquad \text{F.9}$$

The values of $Y(k)$ such that $i_0 \le k < i_1$ form the output of the 1D_FILT$_R$ procedure.

F.3.8.2 Irreversible 1D filtering

The irreversible transformation described in this section is the lifting-based DWT implementation of filtering by the Daubechies 9/7 filter [6].

Equation F.10 describes the 4 "lifting" steps (1 through 4) and the 2 "scaling" steps (5 and 6) of the 1D filtering performed on the extended signal $X_{ext}(n)$ to produce the i_1 coefficients of signal Y.

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

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

$$\begin{cases} Y(2n+1) \leftarrow X_{ext}(2n+1) + (\alpha \times [X_{ext}(2n) + X_{ext}(2n+2)]) & [STEP1] \\ Y(2n) \leftarrow X_{ext}(2n) + (\beta \times [Y(2n-1) + Y(2n+1)]) & [STEP2] \\ Y(2n+1) \leftarrow Y(2n+1) + (\gamma \times [Y(2n) + Y(2n+2)]) & [STEP3] \\ Y(2n) \leftarrow Y(2n) + (\delta \times [Y(2n-1) + Y(2n+1)]) & [STEP4] \\ Y(2n+1) \leftarrow -K \times Y(2n+1) & [STEP5] \\ Y(2n) \leftarrow (1/K) \times Y(2n) & [STEP6] \end{cases} \qquad \text{F.10}$$

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

$$\begin{cases} \alpha = -1{,}586\ 134\ 342 \\ \beta = -0{,}052\ 980\ 118 \\ \gamma = 0{,}882\ 911\ 075 \\ \delta = 0{,}443\ 506\ 852 \end{cases} \qquad \text{F.11}$$

and the scaling factor K is equal to: $K = 1{,}230\ 174\ 105$.

The values of $Y(k)$ such that $i_0 \le k < i_1$ form the output of the 1D_FILTD$_I$ procedure.

Annex G

DC level shifting and component transformations

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

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 component transforms. These component transforms are used to improve compression efficiency. They are not related to colour transforms used to map colour values for display purposes. One component transform 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 component transform.

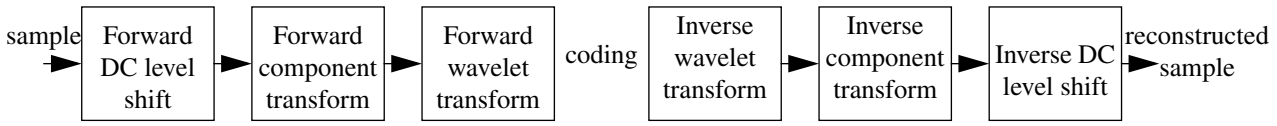


Figure G-1 — Placement of the DC level shifting with component transform

Figure G-2 shows the flow of DC level shifting in the system without a component transform.

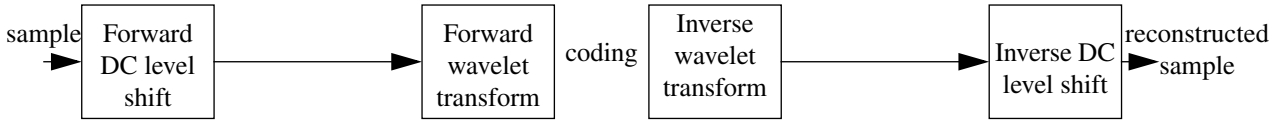


Figure G-2 — Placement of the DC level shifting without component transform

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 either prior to computation of the forward component transform (RCT or ICT), if used. Otherwise it is performed prior to the transform described in Annex F. If the MSB of $Ssiz^i$ from the SIZ marker segment (see Annex 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

$$I'(x, y) \leftarrow I(x, y) - 2^{Ssiz^i - 1}. \qquad \text{G.1}$$

G.1.2 Inverse DC level shifting of tile components (normative)

Inverse DC level shifting is performed on reconstructed samples of components that are unsigned only. It is performed either after to computation of the forward component transform (RCT or ICT), if used. Otherwise it is performed prior to the transform described in Annex F. If the MSB of $Ssiz^i$ from the SIZ marker segment (see Annex A.5.1) is zero, all samples I(x,y) of the ith component are level shifted by adding the same quantity from each sample as follows

$$I(x, y) \leftarrow I(x, y) + 2^{Ssiz^c - 1}. \qquad \text{G.2}$$

NOTE — Due to quantization effects, the reconstructed sample values I(x, y) may exceed the dynamic range of the original tile-component samples values.

G.2 Reversible component transformation (RCT)

The use of the reversible component transformation is signaled in the COD marker segment (see Annex A.6.1). The RCT shall only be used with the 5-3 reversible wavelet transform. The RCT is a decorrelating transformation which is applied to the three first components of an image (indexed as 0, 1 and 2). All three of the components shall have the separation on the reference grid (no sub-sampling) and the same bit-depth. There shall be at least three components if this transform is used.

While the RCT is reversible, it is appropriate for use with lossy compression as well as lossless and progressive lossless to lossy compression.

G.2.1 The Forward RCT (informative)

Prior to applying the Forward RCT, the image component samples are DC level shifted, for unsigned components (see Annex F).

The Forward RCT is applied to all image component samples $I_0(x,y)$, $I_1(x,y)$, $I_2(x,y)$, corresponding to the first, second, and third components, and produces transform samples $Y_0(x,y)$, $Y_1(x,y)$, $Y_2(x,y)$:

$$Y_0(x, y) = \left\lfloor \frac{I_0(x, y) + 2I_1(x, y) + I_2(x, y)}{4} \right\rfloor \quad \text{G.3}$$

$$Y_1(x, y) = I_2(x, y) - I_1(x, y) \quad \text{G.4}$$

$$Y_2(x, y) = I_0(x, y) - I_1(x, y) \quad \text{G.5}$$

Equation G.5 and Equation G.4 results in a numeric precision of Y_1 and Y_2 that is one bit greater than the original components, if I_0, I_1, and I_2 were normalized to the same precision. For reversibility, this precision must be maintained.

G.2.2 The Inverse RCT (normative)

After being inverse transformed as described in Annex F, the following transformation is specified to perform the Inverse RCT:

$$I_1(x, y) = Y_0(x, y) - \left\lfloor \frac{Y_2(x, y) + Y_1(x, y)}{4} \right\rfloor \quad \text{G.6}$$

$$I_0(x, y) = Y_2(x, y) + I_1(x, y) \quad \text{G.7}$$

$$I_2(x, y) = Y_1(x, y) + I_1(x, y) \quad \text{G.8}$$

After applying the Inverse RCT, the image component samples are inverse DC level shifted, for unsigned components.

G.3 Irreversible component transformation (ICT).

This section specifies an irreversible component transformation. The use of the irreversible component transformation is signaled in the COD marker segment (see Annex A.6.1). The ICT shall only be used with the 9-7 irreversible wavelet transform. The ICT is a decorrelating transformation which is applied to the three first components of an image (indexed as 0, 1 and 2). There shall be at least three components if this transform is used. All three of the components shall have the

separation on the reference grid (no sub-sampling) and the same bit-depth. There shall be at least three components if this transform is used.

G.3.1 The Forward ICT (informative)

The Forward ICT is applied to all image component samples $I_0(x,y)$, $I_1(x,y)$, $I_2(x,y)$, corresponding to the first, second, and third component, and produces transform samples $Y_0(x,y)$, $Y_1(x,y)$, $Y_2(x,y)$:

$$Y_0(x, y) = (0, 299) \times I_0(x, y) + (0, 587) \times I_1(x, y) + (0, 144) \times I_2(x, y) \quad \text{G.9}$$

$$Y_1(x, y) = -(0, 168\ 75) \times I_0(x, y) - (0, 331\ 26) \times I_1(x, y) + (0, 5) \times I_2(x, y) \quad \text{G.10}$$

$$Y_2(x, y) = (0, 5) I_0(x, y) - (0.41869) \times I_1(x, y) - (0, 081\ 31) I_2(x, y) \quad \text{G.11}$$

NOTE — If the first three components are Red, Green and Blue components, then the Forward ICT can be seen as an approximation of a YCbCr transformation.

NOTE — The Equation G.9, Equation G.10, and Equation G.11 do not imply a required precision for the irrational numbers.

G.3.2 The Inverse ICT (normative)

After being inverse transformed as described in Annex F, the following transformation is specified to perform the Inverse ICT:

$$I_0(x, y) = Y_0(x, y) + (1, 402) \times Y_2(x, y) \quad \text{G.12}$$

$$I_1(x, y) = Y_0(x, y) - (0, 344\ 13) \times Y_1(x, y) - (0, 714\ 14) \times Y_2(x, y) \quad \text{G.13}$$

$$I_2(x, y) = Y_0(x, y) + (1, 772) \times Y_1(x, y) \quad \text{G.14}$$

The Equation G.12, Equation G.13, and Equation G.14 do not imply a required precision for the irrational numbers. After applying the Inverse ICT, the image component samples are inverse DC level shifted, for unsigned components.

G.4 Chrominance component sub-sampling and the image reference grid (informative)

The relationship between the components and the reference grid is signaled in the SIZ marker (see Annex A.5.1) and described in Annex B.1.

G.4.1 Interpretation of multiple components

The interpretation of multiple components is unspecified within the scope of the codestream. Interpretations, such as multiple colour components, may be supplied by the file format, the application, or other source. Moreover, this standard can accommodate multi-component sources that do not require inter-component decorrelating transforms.

Annex H

Coding of images with Regions of Interest

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

This annex describes the Region of Interest (ROI) technology. An ROI is a part of an image that is 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 Description of the Maxshift method

H.1.1 Encoding (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 Annex H.2 for details on how the mask is generated.

The quantized transform coefficients outside of the ROI mask (to be called background coefficients) are scaled down so that the bits associated with the ROI are placed in higher bit-planes than the background. This means that when the entropy coder encodes the quantized transform coefficients, the bit planes associated with the ROI are coded before the information associated with the background. The scaling value used must be sufficiently large to make the smallest non-zero ROI coefficient, $q_{ROI}(x,y)$, larger than the largest background coefficient, $q_{BG}(x,y)$ (Annex H.1.2).

The method can be described using the following steps:

1) Generate ROI mask, M(x,y) (Annex H.2).
2) Find the scaling value, s (Annex H.1.2).
3) Scale down all background coefficients given by M(x,y) using the scaling value, s (Annex H.2).
4) Write the scaling value, s, into codestream using the RGN marker (Annex A.6.3).

After these four steps the quantized transform coefficients are entropy coded as usual.

After the scaling operation, the number of bit-planes to coded is increased by the Maxshift scaling value.

H.1.2 Selection of scaling value, s, at encoder side (normative)

The scaling value, s, must be chosen so that Equation H.1 holds, where $\max(M_b)$ is the largest number of magnitude bit planes, see Equation E.3, for any background coefficient, $q_{BG}(x,y)$ in any code-block in the current component.

$$s \geq max(M_b) \qquad \text{H.1}$$

This means that the scaling value used will be sufficiently large to make the smallest non-zero ROI coefficient, $q_{ROI}(x,y)$, larger than the largest background coefficient, $q_{BG}(x,y)$. This, in turn, means that after the scaling of the background coefficients, all significant bits associated with the ROI will be in higher bit planes than all the significant bits associated with the background.

H.1.3 Decoding (normative)

At the decoder side the received quantized coefficients are compared to the threshold value 2^s, where s is the ROI scaling value for this component obtained from the RGN marker in the codestream (see Annex A.6.3). All coefficients that are found to be lower than 2^s are known to belong to the background. These coefficients are scaled up by 2^s.

The method can be described using the following steps:

1) Get the scaling value, s, from the RGN marker in the codestream (Annex A.6.3).

2) Compare each quantized transform coefficient q(x,y) to 2^s. If the coefficient is below 2^s scale up the coefficient by 2^s

H.2 Region of interest mask generation

To achieve an ROI with better quality than the rest of the image while maintaining a fair amount of compression, bits need to be saved by sending less information for the background. To do this an ROI mask is calculated. The mask is a bit plane indicating a set of quantized transform coefficients whose coding is sufficient in order for the receiver to reconstruct the desired region with better quality than the background (up to lossless). This mask denotes all coefficients that are needed in order to reconstruct the ROI.

To illustrate the concept of ROI mask generation, let us restrict ourselves to a single ROI and a single image component, and identify the pixels that belong to the ROI in the image domain by a binary mask, M[m,n], where

$$M(x, y) = \begin{cases} 1 & \text{wavelet coefficient (x,y) is needed} \\ 0 & \text{accuracy on (x,y) can be sacrificed without affecting ROI} \end{cases} \quad \text{H.2}$$

M is then bit-wise 1 for all ROI coefficients so that if the first bit of M is 1 then M(x, y) belongs to the first ROI.

The mask is a map of the ROI in the image 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 line by line and then column by column. The mask will then indicate which coefficients are needed at this step so that the inverse transform will reproduce the coefficients of the previous mask.

For example, the last step of the inverse transform is a composition of two sub-bands into one. Then to trace this step backwards, the coefficients in the two sub-bands that are needed, are found. The step before that is a composition of four sub-bands into two. To trace this step backwards, the coefficients in the four sub-bands that are needed to give a perfect reconstruction of the coefficients included in the mask for two sub-bands are found.

All steps are then traced backwards to give the mask. If the coefficients corresponding to the mask are transmitted and received, and the inverse transform calculated on them, the desired ROI will be reconstructed with better quality than the rest of the image (up to lossless if the ROI coefficients were coded 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. Please refer to [23][24][25][26] for more details.

H.2.1 Region of Interest mask generation using the W5X3 filter (informative)

In order to get the optimal set of quantized coefficients to be scaled, the following equations described in this section should be used.

To see what coefficients need to be in the mask, the inverse transform is studied. Equation F.3 and Equation F.4 give the coefficients needed to reconstruct X(2n) and X(2n+1) losslessly. It can immediately that these are L(n), L(n+1), H(n-1), H(n), H(n+1) (see Figure H-1). Hence if X(2n) or 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, related to the origin of the reference grid.

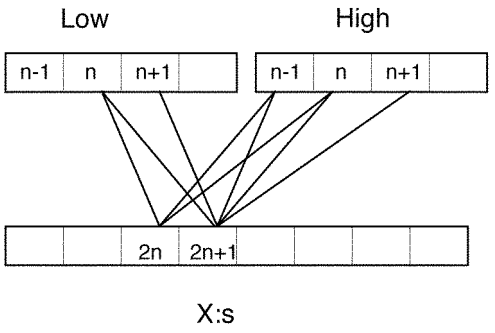


Figure H-1 — The inverse 5x3 transform

H.2.2 Region of Interest mask generation using the W9X7 filter (informative)

Successful decoding does not depend upon the selection of samples to be scaled. In order to get the optimal set of quantized coefficients to be scaled the following equations described in this section should be used.

To see what coefficients need to be in the mask, the inverse transform is studied as in H.2.1. Figure H-2 shows this. Notice that X(2n) and X(2n+1) are even and odd indexed points respectively, related to the related to the origin of the reference grid

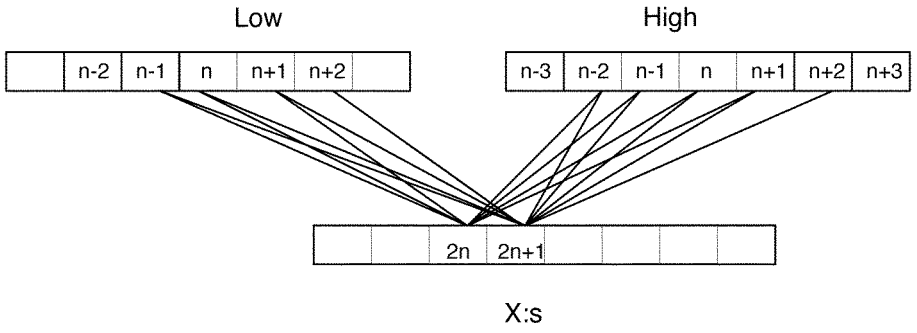


Figure H-2 — The inverse 9x7 transform

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) or X(2n+1) are in the ROI, those Low and High sub-band coefficients are in the mask.

H.3 Remarks on Region of Interest coding

H.3.1 Multi-component remark

For the case of color images, the method applies separately in each color component. If some of the color components are down-sampled, the mask for the down-sampled components is created in the same way as the mask for the non-down-sampled components.

H.3.2 Disjoint regions remark

If the ROI consists of disjoint parts then all parts have the same scaling value s.

H.3.3 Implementation Precision remark

This ROI coding method might in some cases create situations 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 ROI's 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.4 An example of the interpretation 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 in progressive 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 lossless 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 section H.2. 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 having ROI and thus, it is not necessary to wait for the whole 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)

I.1 File format scope

This annex of this Recommendation | International Standard defines an optional file format that applications may choose to use to contain JPEG 2000 compressed image data. While not all applications will use this format, many applications will find that this format meets their needs. However, those applications that do implement this file format shall implement it as described in this entire annex of this Recommendation | International Standard.

This annex of this Recommendation | International Standard

- 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 File format definitions

I.2.1 Glossary

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

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

Box contents: Refers to the data wrapped within the box structure. The contents of a particular box are stored within the DBox field within the Box data structure as defined in Annex I.6

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 as defined in Annex I.6.

Colour component: A component from the codestream that functions as an input to a colour transformation system. For example, a red component or a greyscale component would be a colour component.

Container box: An 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.

JP2 file: The name of file in the file format described in this specification. Structurally, a JP2 file is a contiguous sequence of boxes.

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.

I.2.2 Acronyms

ASCII: American Standard Code for Information Interchange

ICC: International Color Consortium

PCS: Profile Connection Space

UCS: Universal Character Set

URL: Uniform Resource Locator

UTF-8: UCS Transformation Format 8

UUID: Universal Unique Identifier

XML: Extensible Markup Language

I.3 File format normative references

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

- Coded character set—7 bit, American Standard Code for Information Interchange, ANSI X3.4–1986.
- International Color Consortium, ICC profile format specification. ICC.1:1998–09 <http://www.color.org/ICC-1_1998-09.PDF>
- International Electrotechnical Commission. Colour management in multimedia systems: Part 2: Colour Management, Part 2–1: Default RGB colour space—sRGB. IEC 61966–2–1 199x. 9 October 1998 <http://w3.hike.te.chiba-u.ac.jp/IEC/100/ PT61966/parts/> or <http://www.sRGB.com/>.
- W3C, Extensible Markup Language (XML 1.0), Rec-xml-19980210, <http://www.w3.org/TR/REC-xml>
- IETF RFC 2279 UTF–8, A transformation format of ISO 10646. January 1998.
- ISO/IEC 11578:1996 Information technology—Open Systems Interconnection—Remote Procedure Call, <http://www.iso.ch/cate/d2229.html>

I.4 Introduction

The JPEG 2000 file format (JP2 format) provides a foundation for storing application specific data (metadata) in association with a JPEG 2000 codestream, such as that information which is required to display the image. As many applications require a similar set of information to be associated with the compressed image data, it is useful to define the format of that set of data along with the definition of the compression technology and codestream syntax.

Conceptually, the JP2 format encapsulates the JPEG 2000 codestream along with other core pieces of information about that codestream. The building-block of the JP2 format is called an box. All data is encapsulated in boxes. This Recommendation | International Standard defines several types of boxes; the definition of each specific box type defines the kinds of data that may be found within an box of that type. Some boxes will be defined to contain other boxes.

I.4.1 File identification

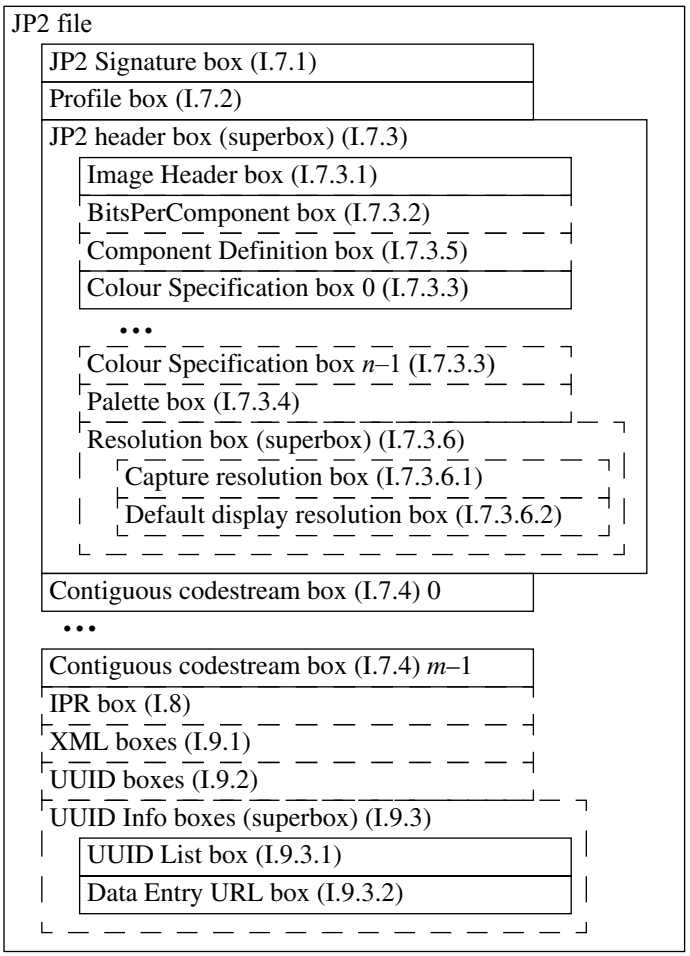
JP2 files can be identified using several mechanisms. When stored in traditional computer file systems, JP2 files should be given the file extension ".jp2" (readers shall also recognize files with the extension ".JP2"). On Macintosh file systems, JP2 files should be given the type code 'jp2\040'.

I.4.2 File organization

A JP2 file represents a collection of boxes. Some of those boxes are independent, and some of those boxes contain other boxes. The binary structure of a file is a contiguous sequence of boxes. The start of the first box shall be the first byte of the file, and the last byte of the last box shall be the last byte of the file.

The binary structure of an box is defined in Annex I.6.

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.

This illustration specifies only the containment relationship between the boxes in the file. A particular order of those boxes in the file is not generally implied. However, the Signature box shall be the first box in a JP2 file and the JP2 header box shall fall before the Contiguous codestream box.

Note that the file is a strict sequence of boxes. Other boxes may be found between the boxes defined in this Recommendation | International Standard. However, all such data shall be in the box format; no other data shall be found in the file.

Figure I-1 — Conceptual structure of a JP2 file

As shown in Figure I-1, a JP2 file contains a JP2 Signature box, JP2 header box, and one or more Contiguous codestream boxes. A JP2 file may also contain other box as determined by the file writer. That JP2 header box contains other boxes, such as the Image Header box and one or more Colour Specification boxes.

I.4.3 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 represents a well-defined colourspace definition. 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 an ICC profile in the file. That profile shall be of either the Monochrome or Three-Channel Matrix-Based class of input profiles as defined by the ICC Profile Format Specification, version 2.2.0. 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 profiles classes. See Annex J.5 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. Note though, that while restricted, these ICC profiles are fully compliant ICC profiles and the image can thus be processed through any ICC compliant engine that supports version 2.2.0 or greater profiles.

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 of 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.4.4 Inclusion of opacity and transparency components

The JP2 file format provides a means to indicate the presence of auxiliary components, such as opacity and transparency, to define the type of those components, and to specify the ordering of all components. When a reader opens the JP2 file, it will determine the ordering and type of each component. The application must then match the component definition and ordering from the JP2 file with the component ordering as defined by the colourspace specification. Once the file components have been mapped to the colour components, the decompressed image can be processed through any needed colourspace transformations.

In many applications, components other than the colour components are required. For example, many images used on web pages contain opacity information; the browser uses this information to blend the image into the background. It is thus desirable to include both the colour and auxiliary components with a single codestream.

I.4.5 Metadata

One important aspect of the JP2 format is the ability to add metadata to a JP2 file. Because all data is encapsulated in boxes, and all boxes have types, the format provides a simple mechanism for a reader to extract 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.4.6 Compliance

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 boxes defined in this Recommendation | International Standard and thus shall correctly interpret all conforming files.

I.5 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 data is properly characterized. The JP2 format provides a multi-level mechanism for characterizing the colourspace of an image. The format also provides a mechanism to specify that an image is not photographic (such as multi-spectral data).

I.5.1 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 and greyscale 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.5.2 Restricted ICC profile method

An application may also specify the colourspace of an image using a restricted set of ICC profiles. This method handles the specification of a 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 image file format, the ICC profile specification defines a type of profile that specifies how samples in a particular colourspace are converted into a standard space (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 does define a specific class of ICC profiles that are easy to implement. The ICC Profile Format Specification defines Monochrome Input and Three-Color Matrix-Based Input Profiles for which the transformation from the source colourspace to the PCS is limited to the application of a non-linearity curve and a 3x3 matrix. It is practical to expect all applications, including simple devices, to be able to process the image through the specified transformation. Thus all conforming applications are required to correctly interpret the colourspace of any image that specifies the colourspace using this subset of possible ICC profile types.

For this Recommendation | International Standard, the class of allowed profiles shall use the XYZ relative version of the PCS.

For the JP2 file format, profiles shall conform to the ICC profile definition as defined by the ICC Profile Format Specification, version 2.2.0, as well as the restrictions specified above. See Annex J.5 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.5.3 Using multiple methods

Architecturally, the format allows for multiple methods to be embedded in a file, providing the reader a choice as to what image processing path should be used to interpret the colourspace of the image. For JP2 files, a conforming reader shall use the first method found in the file (in the first colourspace specification box in the JP2 Header box) and ignore all other methods (found in additional colourspace specification boxes) found in the file.

I.5.4 Palettized images

In addition to specifying the interpretation of the image in terms of colourspace, this Recommendation | International Standard allows for the decoding of single component images 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 a multiple-component codestream.

I.5.5 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 Annex A.5.1 and Annex G). The colourspace transformations specified through the sequence of colour transformation boxes shall be applied to the image samples after the reverse multiple component transformation has been applied to the decompressed samples. While the

application of these decorrelating component transformations is separate, the application of an encoder-based multiple component transformation will often improve the compression of colour image data.

I.6 Box definition

Physically, each object in the file is encapsulated within a binary structure called an box. That binary structure is as follows:

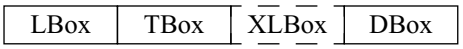


Figure I-2 — Organization of an Box

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 data up to the end of the file. If an 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 other use.

TBox: Box Type. This field specifies the type of data found in the DBox field. The value of this field is encoded as a 32-bit big endian unsigned integer. However, boxes are generally referred to by a ASCII 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 32-bit hexadecimal integers (informative). Also, a space character is shown in the character string translation of the box type as "\040".

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 Data. This field contains the data for the portion of the object contained within this box. The format of that data is dependent on the box type and will be defined individually for each type.

Table I-1 — Binary structure of an box

Field name	Size (bits)	Value
LBox	32	0, 1, 8—(2^{32}–1)
TBox	32	Varies
XLBox	LBox=1, 64 LBox≠1, 0	16—(2^{64}–1) Not applicable
DBox	Varies	Varies

For example, consider the following illustration of a sequence of boxes, including one box that contains other boxes:

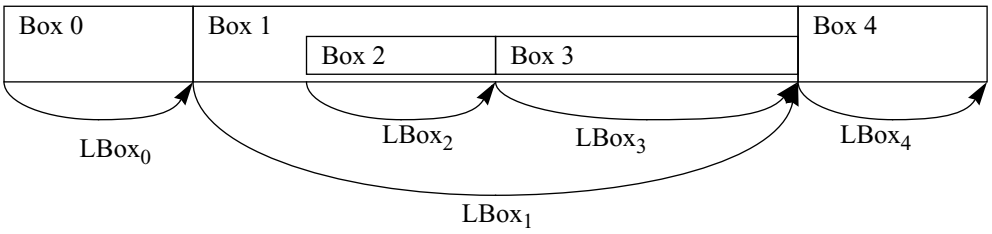


Figure I-3 — Illustration of box lengths

As shown in Figure I-3, the length of each box includes any boxes contained within that box. For example, the length of Box 1 includes the length of Boxes 2 and 3, in addition to the LBox and TBox fields for Box 1 itself. In this case, if the type of Box 1 was not understood by a reader, it would not recognize the existence of boxes 2 and 3 because they would be completely skipped by jumping the length of box 2 from the beginning of box 2.

The following table lists all boxes defined by this Recommendation | International Standard. Indentation within the table indicates the hierarchical containment structure of the boxes within a JP2 file:

Table I-2 — Boxes defined within this Recommendation | International Standard

Box name	Type	Container box	Required?	Notes
JP2 Signature box	‘jP\032\032’ (X‘6A501A1A’)	No	Required	This box uniquely identifies the file as a JP2 file.
Profile box	‘prfl’ (X‘7072666C’)	No	Required	This box specifies profile and compatibility information
JP2 Header box	‘jp2h’ (X‘6A703268’)	Yes	Required	This box contains a series of boxes that contain header-type information about the file.
Image Header box	‘ihdr’ (X‘69686472’)	No	Required	This box contains the size of the image and other related fields.
BitsPerComponent box	‘bpcc’ (X‘62706363’)	No	Optional	This box specifies the bit depth of the components in the file in cases where the bit depth is not constant across all components.
Colour Specification	‘colr’ (X‘636F6C72’)	No	Required	This box specifies the colourspace of the image.
Palette	‘pclr’ (X‘70636C72’)	No	Optional	This box specifies the palette which maps a single component in index space to a multiple-component image.
Component Definition box	‘cdef’ (X‘63646566’)	No	Optional	This box specifies the type and ordering of the components within the codestream.
Resolution box	‘res ’ (X‘72657320’)	Yes	Optional	This box specifies the resolution of the image.

Table I-2 — Boxes defined within this Recommendation | International Standard

Box name	Type	Container box	Required?	Notes
Capture resolution box	'resc' (X'72657363')	No	Optional	This box specifies the resolution at which the image was captured.
Default Display resolution box	'resd' (X'72657364')	No	Optional	This box specifies the default resolution at which the image should be displayed.
Contiguous Codestream boxes	'jp2c' (X'6A703263')	No	Required	This box contains the codestream as defined by Annex A of this Recommendation	International Standard
Intellectual Property box	'jp2i' (X'6A703269')	No	Optional	This box contains intellectual property information about the image.
XML box	'xml\040' (X'786D6C20')	No	Optional	This box provides a tool by which vendors can add XML formatted information to a JP2 file.
UUID box	'uuid' (X '75756964')	No	Optional	This box provides a tool by which vendors can add additional data to a file without risking conflict with other vendors.
UUID Info box	'uinf' (X'75696E66')	Yes	Optional	This box provides a tool by which a vendor may provide access to additional information associated with a UUID
UUID list box	'ulst' (X'75637374')	No	Optional	This box specifies a list of UUID's.
URL box	'url\040' (X'75726C20')	No	Optional	This box specifies a URL.

I.7 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 Annex I.6. The following sections 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.6.

I.7.1 JP2 Signature box

The JP2 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 JP2 signature box shall be the first box in the file, and all files shall contain one and only one JP2 signature box.

The type of the JP2 signature box shall be 'jP\032\032' (X'6A501A1A'). The length of this box shall be 12 bytes. The contents of this box shall be the 4-byte character string '<CR><LF><X'87'><LF>' (X'0D0A870A'). For file verification purposes, this box can be considered a fixed-length 12-byte string which shall have the value:
X'0000 000C 6A50 1A1A 0D0A 870A'.

The combination of the particular type and contents for this box enable an application to detect a common set of file transmission errors. The CR-LF sequence in the contents catches bad file transfers that alter newline sequences. The control-Z character in the type stops file display under MS-DOS. The final linefeed checks for the inverse of the CR-LF translation problem. The third character of the box contents has its high-bit set to catch bad file transfers that clear bit 7.

I.7.2 Profile box

The Profile box specifies information about the Recommendations | International Standards with which the file is compatible, and allows the file creator to specify the Recommendations | International Standards representing the intended purpose of the file. This box shall immediately follow the JP2 signature box. Also, all files shall contain one and only one Profile box.

The type of the Profile Box shall be ‘prfl’ (X‘7072666C’). The contents of this box shall be as follows:

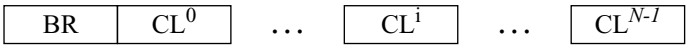


Figure I-4 — Organization of the contents of a Profile box

BR: Brand. This field specifies the governing Recommendation | International Standard on which the file is based. This field is specified by a four byte string of ASCII characters. The value of this field for files governed by this Recommendation | International Standard shall be ‘jp2\040’.

This field only describes the governing Recommendation | International Standard for the file. Readers must examine the CL^i fields to determine if they can properly interpret the file.

Other values of the Brand field are reserved for ISO use.

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 ASCII characters. A file that conforms to this Recommendation | International Standard shall have at least one CL^i field in the Profile box, and shall contain the value ‘jp2\040’ in one of the CL^i fields in the Profile box.

The number of CL^i fields is determined by the length of this box.

Table I-3 — Format of the contents of the Profile box

Field name	Size (bits)	Value
BR	32	0—(2^{32}–1)
CL^i	32	0—(2^{32}–1)

I.7.3 JP2 header box (superbox)

The JP2 header box contains generic information about the file, such as number of samples, colourspace, and resolution. This box is a superbox. The format of the Profile box is as follows:

Within a JP2 file (considered as a superbox), there shall be one and only one JP2 header box. The JP2 header box may be located anywhere within the file after the JP2 signature box but before the contiguous codestream box. It also must be at the same level as the JP2 signature box (it shall not be inside any other superbox within the file).

The type of the JP2 header box shall be ‘jp2h’ (X‘6A703268’).

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 follows:

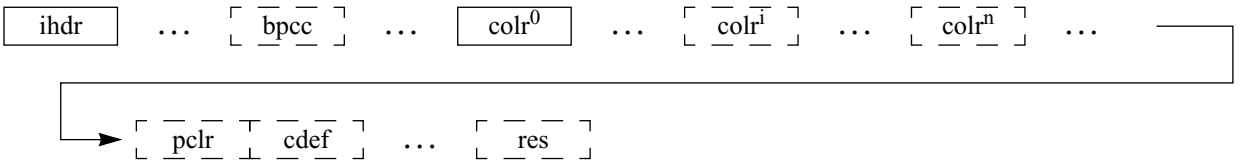


Figure I-5 — Organization of the contents of a JP2 header box

ihdr: Image Header Box. This box specifies information about the image, such as its height and width. Its structure is specified in Annex I.7.3.1. This box shall be the first box in the JP2 header box.

bpcc: BitsPerComponent box. This box specifies the bit depth of each component in the codestream after decompression. Its structure is specified in Annex I.7.3.2. 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 Annex I.7.3.3. 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.

pclr: Palette box. This box defines the palette to use to create multiple components from a single component. Its structure is specified in Annex I.7.3.4. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

cdef: Component Definition box. This box defines the components in the codestream. Its structure is specified in Annex I.7.3.5. 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 resolutions of the image. Its structure is specified in Annex I.7.3.6. This box may be found anywhere in the JP2 header box provided that it comes after the Image Header box.

I.7.3.1 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 24 bytes, including the box length and type fields. Note that much of the information within the Image Header box is redundant with information stored in the codestream itself.

The type of the Image Header box shall be 'ihdr' (X'69686472') and contents of the box shall have the following format:

VERS	NC	HEIGHT	WIDTH	BPC	C	UnkC	IPR

Figure I-6 — Organization of the contents of an Image Header box

VERS: Version. This parameter defines the version number of this JP2 specification for which the file complies. The parameter is defined as a 2-byte big endian unsigned integer with the most significant byte containing the major version number (currently defined as 1) and the least significant byte containing a minor revision number (currently defined as 0).

The value of this field is X'0100.'

A major version number increment (if there ever is one) represents an incompatible change in JP2 files. Decoders should give up if they encounter an unrecognized major version number. Minor version

number increments represent backward compatible changes. Decoders should continue to process JP2 files even if the minor version number is unrecognized.

NC: Number of components. This parameter specifies the number of components in the image and is stored as a 2-byte big endian unsigned integer.

HEIGHT: Image height. The value of this parameter indicates the number of lines of the rendered image. If the file contains only one codestream, then this value shall be the same as the value of the Ysiz parameter in the SIZ marker segment in that codestream. Otherwise, this field specifies the height of the image into which the sequence of codestreams are rendered. This field is stored as a 4-byte big endian unsigned integer.

WIDTH: Image width. The value of this parameter indicates the number of samples per line of the rendered image. If the file contains only one codestream, then this value shall be the same as the value of the Xsiz parameter in the SIZ marker segment in that codestream. Otherwise, this field specifies the width of the image into which the sequence of codestreams are rendered. This field is stored as a 4-byte big endian unsigned integer.

BPC: Bits per component. This parameter specifies the bit depth of the components in the image and is stored as a 1-byte field.

If the bit depth is the same for all components, then this parameter specifies the actual bit depth. If the components vary in bit depth, then the value of this field shall be zero and the JP2 header box shall also contain a BitsPerComponent box defining the bit depth of each component (as defined in Annex I.7.3.2).

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. If the value of this field is not 7, then this file is not a conforming JP2 file.

UnkC: Colourspace Unknown. This field specifies if the actual colourspace of the image data 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 the colourspace 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 other use.

IPR: Intellectual Property. This parameter 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 Annex I.8. Other values are reserved for ISO use.

Table I-4 — Format of the contents of the Image Header box

Field name	Size (bits)	Value
VERS	16	X‘0100’
NC	16	1—(2^{16}–1)
HEIGHT	32	1—(2^{32}–1)
WIDTH	32	1—(2^{32}–1)
BPC	8	-127—127

Table I-4 — Format of the contents of the Image Header box

Field name	Size (bits)	Value
C	8	7
Unk	8	0—1
IPR	8	0—1

I.7.3.2 BitsPerComponent box

The BitsPerComponent box specifies the bit depth of each component. If the bit depth is constant across all components in the codestream, then this box shall not be found. Otherwise, this box specifies the bit depth of each component. The order of bit depth values in this box is the actual order those components are enumerated within the codestream. The exact location of this box within the JP2 header box may vary provided that it follows the Image Header box.

The type of the BitsPerComponent Box shall be 'bpcc' (X'62706363'). The contents of this box shall be as follows:

Figure I-7 — Organization of the contents of a BitsPerComponent box

BPC^i: Bits per component. This parameter specifies the bit depth of component *i*, encoded as a 1-byte ones-complement integer. The ordering of the components within the BitsPerComponent Box shall be the same as the ordering of the components within the codestream. The number of BPC^i fields shall be the same as the value of the NC field from the Image Header box.

The low 7-bits of the value indicate the bit depth of this component. The high-bit indicates whether the component is signed or unsigned. If the high-bit is 1, then the component contains signed values. If the high-bit is 0, then the component contains unsigned values.

Table I-5 — Format of the contents of the BitsPerComponent box

Field name	Size (bits)	Value
BPC^i	8	-127— -1, 1—127

I.7.3.3 Colour Specification box

Each Colour Specification box defines one method by which an application can interpret the colourspace of the decompressed image data. A JP2 file may contain multiple Colour Specification boxes, specifying different methods for achieving "equivalent" results. Note that 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 data. A conforming JP2 shall ignore all Colour Specification boxes after the first.

The type of a Colour Specification box shall be 'colr' (X'636F6C72'). The contents of a Colour Specification box is as follows:

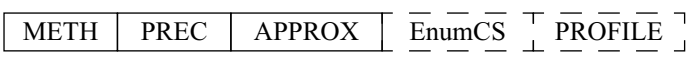


Figure I-8 — Organization of the contents of a Colour Specification box

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. Legal values of this field are as follows:

Table I-6 — 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 a Restricted ICC profile in the PROFILE field. This profile specifies the transformation needed to convert the decompressed image data into the PCS. If the value of METH is 2, then the ICC profile shall conform to the definition of either a Monochrome Input Profile or a Three-Component Matrix-Based Input Profile as defined in the ICC profile specification, version 2.2.0. In addition, the value of the Profile Connection Space field in the profile header in the embedded profile shall be 'XYZ ' (X'58595A20') indicating that the output colourspace of the profile is XYZ data. Note that 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 the JP2 file format, profiles shall conform to the ICC profile definition as defined by the ICC Profile Format Specification, version 2.2.0, as well as the restrictions specified above. See Annex J.5 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. Those fields shall be ignored.

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 and 17 as defined in Table I-7:

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

Table I-7 — 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 Eqs. (2) through (4) of IEC 61966–2–1 (sRGB) specification: $Y' = Y_{8bit}/255$ I.1 $for(Y' \le 0.04045),\ Y_{lin} = Y'/12.92$ $for(Y' > 0.04045),\ Y_{lin} = \left(\frac{Y' + 0.055}{1.055}\right)^{2.4}$ I.2 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.
other values	Reserved for other ISO uses

profile shall conform to the Monochrome Input Profile class or the Three-Component Matrix-Based Input Profile class as defined in the ICC profile specification.

Table I-8 — Format of the contents of the Colr box

Field name	Size (bits)	Value
METH	8	1—2
PREC	8	0
APPROX	8	0
EnumCS	32 if METH=1 0 if METH=2	0—(2^{32}–1) no value
PROFILE	Varies	Varies

I.7.3.4 Palette box

The colour palette specified in this box is applied to the single colour component to convert the single value to a tuple. The colourspace of the generated tuple is then interpreted based on the values of the colour specification boxes in the JP2 Header box in the file.

The type of the palettized colour box shall be ‘pclr’ (X‘70636C72’). The contents of this box shall be as follows:

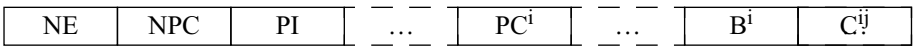


Figure I-9 — Organization of the contents of the Palette box

NE: Number of entries in the table. This value shall be in the range 1 to 1024.

NPC: Number of components created by the application of the palette. For example, if the palette turns a single index component into a three-component RGB images, then the value of this field shall be 3.

PI: Palette input. This field specifies the number of the component that should be used as the input to the palette (the index component). This field is encoded as a 2 byte unsigned integer, and the value of this field shall be less than the number of components specified by the NC field in the Image Header box

PC^i: Component number of palette created component *i*. This field specifies a number by which the component *i* of the palette table shall be referred. These values will be used by the Component Definition box to specify the individual components of the palette. This value shall be greater than the number of components specified in the Image Header Box, and shall not be the same as the value of any other PC^i field in this box. The number of PC^i fields shall be the same as the value of the NPC field.

B^i: This parameter specifies the bit depth of generated component *i*, encoded as a 8-bit integer. 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. The number of B^i values shall be the same as the value of the NPC field.

C^{ij}: The generated component value for entry *j* for component *i*. C^{ij} values are organized in component major order; all of the component values for entry *j* are grouped together, followed by all of the entries for component *j*+1. The size of C^{ij} is the value specified by field B^i. The number of components shall be the same as the NPC field. The number of C^{ij} values shall be the number of created components (the NPC field) x the number of entries in the palette (NE).

Table I-9 — Format of the contents of the Palette box

Field name	Size (bits)	Value
NE	16	1—1024
NPC	8	1—255
PI	16	0—(2^{16}–1)
PC^i	16	0—(2^{16}–1)
B^i	8	-127— -1, 1—127
C^{ij}	Varies	Varies

I.7.3.5 Component Definition box

The component definition box specifies the meaning of the data in each component in the codestream. The exact location of this box within the JP2 header box may vary provided that it follows the Image Header box.

This box contains an array of component descriptions. For each description, three values are specified: the number of the component described by that association, the type of that component, and the association of that component with particular colours. This box may specify multiple descriptions for a single component; however, the type value in each description for the same component shall be the same in all descriptions.

If the codestream contains only colour components and those components are ordered in the same order as the associated colours (for example, an RGB images with three components in the order R, G, then B), then this box shall not exist. If there are any auxiliary components or the components are not in the same order as the colour numbers, then the Component Definition box shall be found within the JP2 header box with a complete list of component definitions. However, if this file contains a Palette box, the component specified as input to the palette (in the PI field) shall not be listed in the Component Definition box.

If a multiple component transform is specified within the codestream, the component ordering box shall specify the existence of red, green and blue colours as components 0, 1 and 2 in the codestream, respectively.

The type of the Component Definition box shall be ‘cdef’ (X‘63646566’). The contents of this box shall be as follows:

Figure I-10 — Organization of the contents of a Component Definition box

N: Number of component descriptions. This field specifies the number of component descriptions in this box. This field is encoded as a 2-byte big endian unsigned integer.

Cn^i: Component number. This field specifies the number of the component for this description. The value of this field represents the number of the component as defined within the codestream or created by the application of a palette to a single component codestream. The numbers of components created by the application of the palette are defined by the Palette box. This field is encoded as a 2-byte big endian unsigned integer.

Typ^i: Component type. This field specifies the type of the component for this description. The value of this field represents the type of data contained within the component. This field is encoded as a 2-byte big endian unsigned integer. Legal values of this field are as follows:

Table I-10 — Typ^i field values

Value	Meaning
0	This component is the colour component for the associated colour
1	Opacity. A sample value of 0 indicates that the sample is 100% transparent, and the maximum value of the component (related to the bit depth of the component) indicates a 100% opaque sample.
2	Premultiplied opacity. An opacity component as specified above, except that the value of the opacity component has been multiplied into the colour components for which this component 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 premultiplied sample (the sample stored in the image, α is the value of the opacity component, and α_{max} is the maximum value of the opacity component as defined by the bit depth of the opacity component.
3—(2^{16}–2)	Reserved for ISO use
2^{16}–1	The type of this component is not specified

Asoc^i: Component association. This field specifies the number of the colour for which this component is directly associated (or a special value to indicate the whole image or the lack of an association). For example, if this component is an opacity blending component for the red component in an RGB colourspace, this field would specify the number of the colour red. Table I-11 specifies legal association values. Table I-12 specifies legal colour numbers. This field is encoded as a 2-byte big endian unsigned integer.

Table I-11 — Asoc^i field values

Value	Meaning
0	This component is associated as the image as a whole (for example, a component independent opacity blending channel
1—(2^{16}–2)	This component is associated with the a particular colour as indicated by this value. This value is used to associate a particular component with a particular aspect of the specification of the colourspace of this image. For example, indicating that a component is associated with the red component of an RGB image allows the reader to associate that decoded component with the Red input to an ICC profile contained within a Colour Specification box. Colour indicators are specified in Table I-12
2^{16}–1	This component is not associated with any particular colour

Table I-12 — 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			
The following colourspace classes are listed for future reference, as well as to aid in understanding of the use of the Asoc^i field				
XYZ	X	Y	Z	
Lab	L	a	b	
Luv	L	u	v	
YC_bC_r	Y	C_b	C_r	
Yxy	Y	x	y	
HSV	H	S	V	
HLS	H	L	S	
CMYK	C	M	Y	K
CMY	C	M	Y	
Jab	J	a	b	
n colour colourspaces	1	2	3	4

In this box, component numbers refer to the number of that particular component within the codestream. Colour numbers specify how that component shall be interpreted based on the specification of the colourspace of the image.

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 number of that component in the codestream (either directly or as generated by applying the reverse multiple component transform). Applications that are only concerned with extracting the colour components can treat the Typ/Asoc field pair

as a four-byte value where the combined value maps directly to the colour numbers (as the Typ field for a colour component shall be 0).

In another example, the codestream may contain a component *i* that specifies opacity blending data for the red and green components, and a component *j* that specifies opacity blending data for the blue component. In that file, the following {Cn, Typ, Asoc} tuples would be found in the Component Definition box: {*i*, 1, 1}, {*i*, 1, 2} and {*j*, 1, 3}.

There shall not be more than one component 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 component, and a greyscale image shall contain only one grey component. There also shall not be more than one opacity component associated with a single colour component in an image.

Table I-13 — Component definition & ordering data structure values

Parameter	Size (bits)	Value
N	16	0—(2^{16}–1)
Cn^i	16	0—(2^{16}–1)
Typ^i	16	0—(2^{16}–1)
$Asoc^i$	16	0—(2^{16}–1)

I.7.3.6 Resolution box (superbox)

This box specifies the capture and default display resolution of this image. If this box exists, it shall contain either a capture display resolution box, or a default display resolution box, or both.

The type of a Resolution box shall be ‘res ’ (X‘72657320’). The contents of the resolution box are as follows:

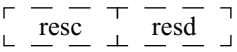


Figure I-11 — Organization of the contents of the Resolution box

resc: Capture resolution box. This box specifies the resolution at which this image was captured. The format of this box is specified in Annex I.7.3.6.1.

resd: Default display resolution box. This box specifies the default resolution at which this image should be displayed. The format of this box is specified in Annex I.7.3.6.2

I.7.3.6.1 Capture resolution box

This box specifies the resolution at which the source was digitized to create the image samples specified by the codestream. For example, this may specify the resolution of the flatbed scanner that captured a page from a book. The capture resolution could also specify the resolution of an aerial digital camera or satellite camera.

The vertical and horizontal capture resolutions are calculated using the six parameters (Table I-14) stored in this box in the following two equations, respectively:

$$VRc = \frac{VRcN}{VRcD} \times 10^{VRcE} \qquad \text{I.4}$$

$$HRc = \frac{HRcN}{HRcD} \times 10^{HRcE} \qquad \text{I.5}$$

The values *VRc* and *HRc* are always in samples/meter. If an application requires the resolution in another unit, then that application must apply the appropriate conversion.

The type of a Capture resolution box shall be 'resc' (X'72657363'). The contents of the Capture resolution box are as follows:

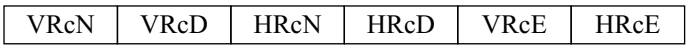


Figure I-12 — Organization of the contents of the Capture Resolution box

VRcN: Vertical Capture resolution numerator. This parameter specifies the *VRcN* value in Equation I.4, which is used to calculate the vertical capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRcD: Vertical Capture resolution denominator. This parameter specifies the *VRcD* value in Equation I.4, which is used to calculate the vertical capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRcN: Horizontal Capture resolution numerator. This parameter specifies the *HRcN* value in Equation I.5, which is used to calculate the horizontal capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRcD: Horizontal Capture resolution denominator. This parameter specifies the *HRcD* value in Equation I.5, which is used to calculate the horizontal capture resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRcE: Vertical Capture resolution exponent. This parameter specifies the *VRcE* value in Equation I.4, which is used to calculate the vertical capture resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

HRcE: Horizontal Capture resolution exponent. This parameter specifies the *HRcE* value in Equation I.5, which is used to calculate the horizontal capture resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

Table I-14 — Format of the contents of the Capture resolution box

Field name	Size (bits)	Value
VRcN	16	1—(2^{16}–1)
VRcD	16	1—(2^{16}–1)
HRcN	16	1—(2^{16}–1)
HRcD	16	1—(2^{16}–1)
VRcE	8	-128—127
HRcE	8	-128—127

I.7.3.6.2 Default display resolution box

This box specifies a default resolution at which the image should be displayed. For example, this may be used to determine the size of the image on a page when the image is placed in a page-layout program. Note, however, that this value is only a default. Each application must determine an appropriate display size for that application.

The vertical and horizontal display resolutions are calculated using the six parameters (Table I-15) 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 samples/meter. If an application requires the resolution in another unit, then that application must apply the appropriate conversion.

The type of a Default display resolution box shall be ‘resd’ (X‘72657364’). The contents of the Default display resolution box are as follows:

VRdN	VRdD	HRdN	HRdD	VRdE	HRdE

Figure I-13 — Organization of the contents of the Default Display Resolution box

VRdN: Vertical Display resolution numerator. This parameter specifies the *VRdN* value in Equation I.6, which is used to calculate the vertical display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRdD: Vertical Display resolution denominator. This parameter specifies the *VRdD* value in Equation I.6, which is used to calculate the vertical display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRdN: Horizontal Display resolution numerator. This parameter specifies the *HRdN* value in Equation I.7, which is used to calculate the horizontal display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

HRdD: Horizontal Display resolution denominator. This parameter specifies the *HRdD* value in Equation I.7, which is used to calculate the horizontal display resolution. This parameter is encoded as a 16-bit big endian unsigned integer.

VRdE: Vertical Display resolution exponent. This parameter specifies the *VRdE* value in Equation I.6, which is used to calculate the vertical display resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

HRdE: Horizontal Display resolution exponent. This parameter specifies the *HRdE* value in Equation I.7, which is used to calculate the horizontal display resolution. This parameter is encoded as a twos-compliment 8-bit signed integer.

Table I-15 — Format of the contents of the Default display resolution box

Field name	Size (bits)	Value
VRdN	16	1—(2^{16}–1)
VRdD	16	1—(2^{16}–1)
HRdN	16	1—(2^{16}–1)
HRdD	16	1—(2^{16}–1)
VRdE	8	-128—127
HRdE	8	-128—127

I.7.4 Contiguous codestream box

The Contiguous codestream box contains a valid and complete JPEG 2000 codestream, as defined in Annex A of this Recommendation | International Standard. When displaying the image, a conforming reader shall ignore all codestreams after the first codestream found in the file.

The type of a contiguous codestream box shall be 'jp2c' (X'6A703263'). The contents of the box shall be as follows:

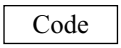


Figure I-14 — Organization of the contents of the Contiguous codestream box

Code: This field contains a valid and complete JPEG 2000 codestream as specified by Annex A of this Recommendation | International Standard.

Table I-16 — Format of the contents of the Contiguous codestream box

Field name	Size (bits)	Value
Code	Varies	Varies

I.8 Adding intellectual property rights information in JP2

This Recommendation | International Standard specifies an box type for an 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 data is beyond the scope of this Recommendation | International Standard.

The type of the Intellectual Property Box shall be 'jp2i' (X'6A703269').

I.9 Adding vendor specific information to the JP2 file format

The following boxes provide a set of tools by which applications can add vendor specific information to the JP2 file format. All of the following boxes are optional in conforming files and may be ignored by conforming readers.

I.9.1 XML boxes

An XML box contains vendor specific data (in XML format) other than that data defined within this Recommendation | International Standard. There may be multiple XML boxes within the file, and those boxes may be found anywhere in the file except before the JP2 signature box.

The type of an XML box is 'xml\040' (X'786D6C20'). The contents of the box shall be as follows:

Figure I-15 — Organization of the contents of the XML box

DATA: This field shall be valid XML as defined by REC-xml-19980210.

The existence of any XML boxes is optional for conforming files. Also, any XML box shall not contain any information necessary for decoding the image to the extent that is defined within this part of this Recommendation | International Standard, and the correct interpretation of the data in any XML box shall not change the visual appearance of the image. All readers may ignore any XML box in the file.

I.9.2 UUID boxes

A UUID box contains vendor specific data other than that data defined within this Recommendation | International Standard. There may be multiple UUID boxes within the file, and those boxes may be found anywhere in the file except before the JP2 signature box.

The type of a UUID box shall be 'uuid' (X'75756964'). The contents of the box shall be as follows:

Figure I-16 — Organization of the contents of the UUID box

ID: This field contains a 16-byte UUID as specified by ISO/IEC 11578:1996. The value of this UUID specifies the format of the vendor specific data stored in the DATA field and the interpretation of that data.

DATA: This field contains the vendor specific data. The format of this data is defined outside of the scope of this standard, but is indicated by the value of the UUID field.

Table I-17 — Format of the contents of a UUID box

Field name	Size (bits)	Value
UUID	128	Varies
DATA	Varies	Varies

The existence of any UUID boxes is optional for conforming files. Also, any UUID box shall not contain any information necessary for decoding the image to the extent that is defined within this part of this Recommendation | International Standard, and the interpretation of the data in any UUID box shall not change the visual appearance of the image. All readers may ignore any UUID box.

I.9.3 UUID Info boxes (superbox)

While it is useful to allow vendors to extend JP2 files by adding binary data using UUID boxes, it is also useful to provide information in a standard form which can be used by non-extended applications to get more information about the extensions in the file. This information is contained in UUID Info boxes. A JP2 file may contain zero or more UUID Info boxes. These boxes may be found anywhere in the top level of the file (the superbox of a UUID Info box shall be the JP2 file itself) except before the signature box.

Note that these boxes, if present, may not provide a complete index for the UUID's in the file, may reference UUID's not used in the file, and possibly may provide multiple references for the same UUID.

The type of a UUID Info box shall be 'uinf' (X'75696E66'). The contents of a UUID Info box are as follows:

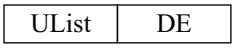


Figure I-17 — Organization of the contents of a UUID Info box

UList: UUID List box. This box contains a list of UUID's for which this UUID Info box specifies a link to more information. The format of the UUID List box is specified in Annex I.9.3.1.

DE: Data Entry URL box. This box contains a URL. An application can acquire more information about the UUID's contained in the UUID list box. The format of a Data Entry URL box is specified in Annex I.9.3.2

I.9.3.1 UUID List box

This box contains a list of UUID's. The type of a UUID List box shall be 'ulst' (X'75637374'). The contents of a UUID List box shall be as follows:

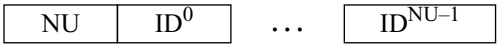


Figure I-18 — Organization of the contents of a UUID Info box

NU: Number of UUID's. This field specifies the number of UUID's found in this UUID List box. This field is encoded as a 16-bit big endian unsigned integer.

ID^i: ID. This field specifies one UUID, as specified in ISO/IEC 11578:1996, 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-18 — UUID List box contents data structure values

Parameter	Size (bits)	Value
NU	16	0—(2^{16}–1)
UUID^i	128	0—(2^{128}–1)

I.9.3.2 Data Entry URL box

This box contains a URL which can use used by an application to acquire more information about the associated vendor specific extensions. The format of the data acquired through the use of this URL is not defined in this Recommendation | International Standard. The URL type should be of a service which delivers a file (e.g. URL's of type file, http, ftp, etc.), which ideally also permits random access. Relative URL's are permissible and are relative to the file containing this data reference.

The type of a Data Entry URL box shall be 'url\040' (X'75726C20'). The contents of a Data Entry URL box shall be as follows:

Figure I-19 — Organization of the contents of a URL box

VERS: Version number. This field specifies the version number of the format of this box. The value of this field shall be 0.

FLAG: Flags. This field is reserved for other use to flag particular attributes of this box. The value of this field shall be 0.

LOC: Location. This field specifies the URL of the additional information associated with the UUID's contained in the UUID List box within the same UUID Info superbox. The URL is encoded as a null terminated string of UTF-8 characters

Table I-19 — URL box contents data structure values

Parameter	Size (bits)	Value
VERS	8	0
FLAG	24	0
LOC	varies	varies

I.10 Dealing with unknown boxes

A valid codestream may contain boxes not known to applications based solely on this Recommendation | International Standard. If a conforming reader finds an box that it does not understand, it shall skip and ignore that box.

Annex J

Examples and Guidelines

This Annex includes a number of examples intended to indicate how the encoding process works, and how the resulting data stream should be output. This Annex is entirely informative.

J.1 Software Conventions Adaptive Entropy Decoder

This annex provides some alternative flowcharts for a version of the adaptive entropy decoder. This alternative version may be more efficient when implemented in software, as it has fewer operations along the fast path. This annex is strictly informative.

The alternative version is obtained by making the following substitutions.

Replace the flowchart in Figure 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.

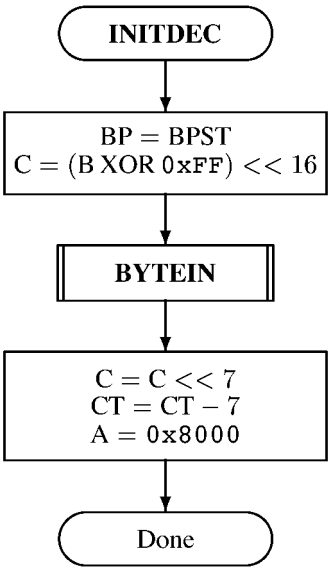


Figure J-1 — Initialisation of the software-conventions decoder

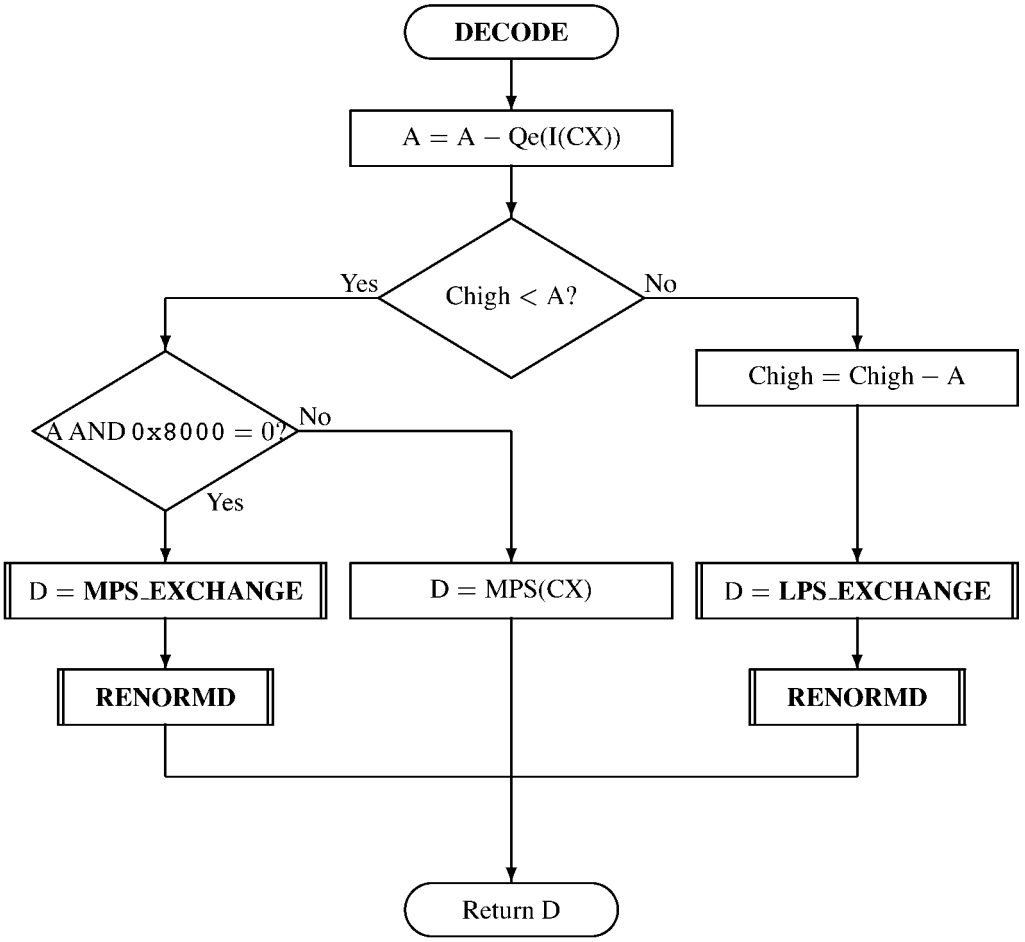


Figure J-2 — Decoding an MPS or an LPS in the software-conventions decoder

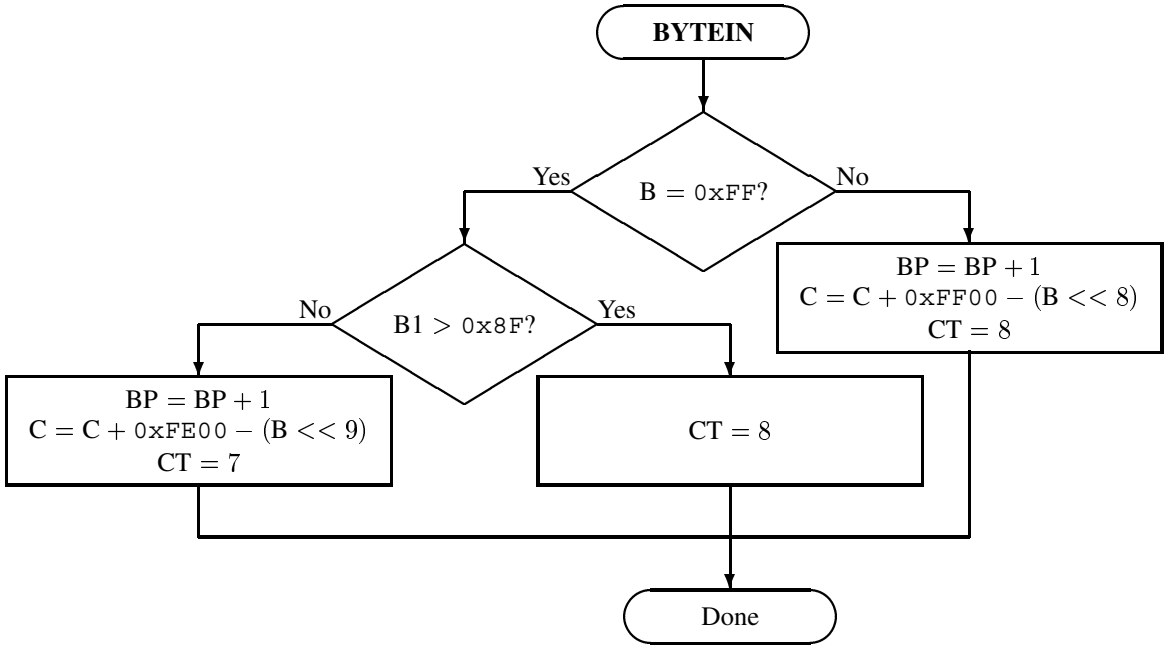


Figure J-3 — Inserting a new byte into the C register in the software-conventions decoder

J.2 Row-based wavelet transform

Described here is an example of a row-based wavelet transform for the 9-7 filter well suited for compression devices which received and transferred image data in a serial manner. Traditional wavelet transform implementations require the whole image to be buffered and filtering to be performed in vertical and horizontal directions. While filtering in the horizontal direction is very simple, filtering in the vertical direction is more involved. Filtering along a row requires one row to be read; filtering along a column requires the whole image to be read. This explains the huge bandwidth requirements of the traditional wavelet transform implementation. The row-based wavelet transform overcomes the previous limitation while providing the exact same transformed coefficients as traditional wavelet transform implementation. However, the row-based wavelet transform alone does not provide a complete row-based encoding paradigm. A complete row-based coder has to take also into account all the following coding stage up to the entropy coding stage.

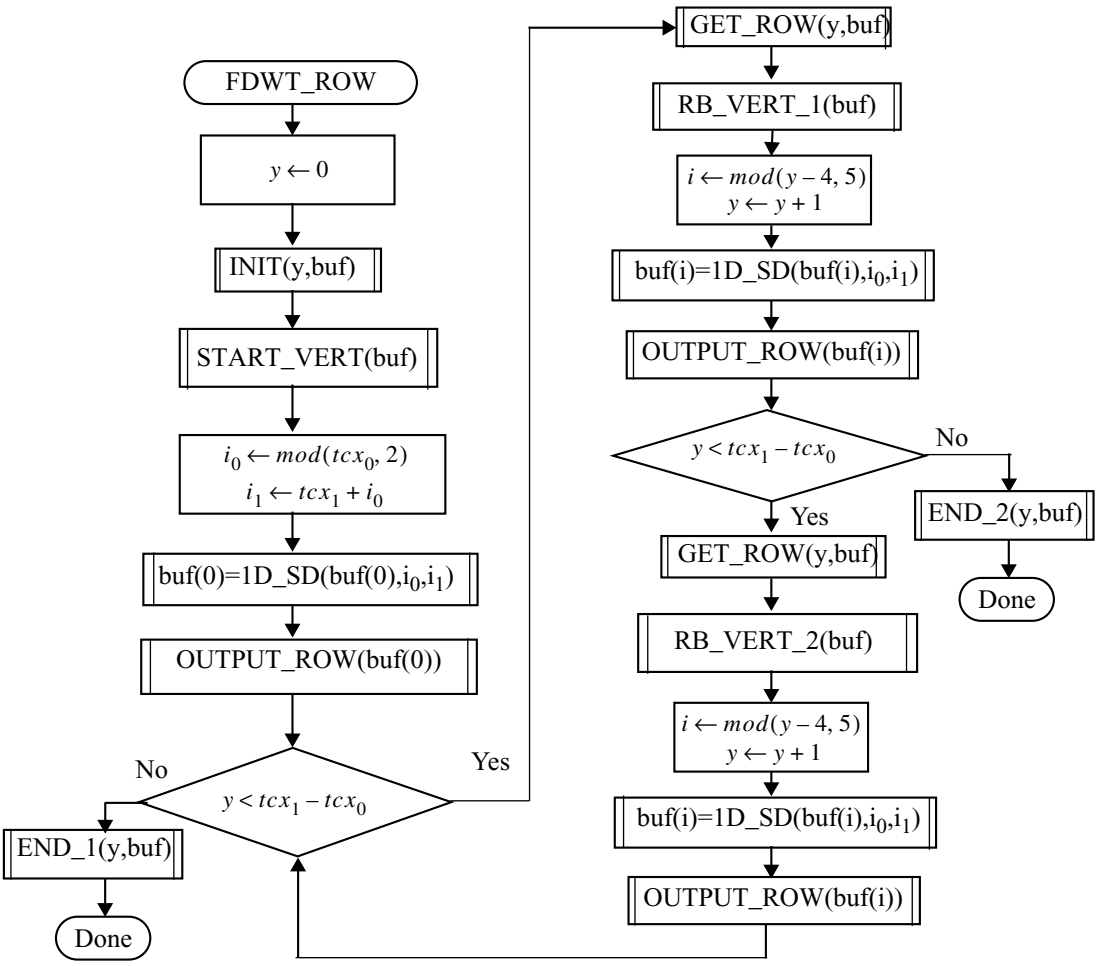


Figure J-4 — The FDWT_ROW procedure

J.2.1 The FDWT_ROW procedure

The FDWT_ROW procedure uses one buffer buf(i,j) of five lines, $0 \le i \le 4$, for performing a one level wavelet decomposition on one row of length tcy_1-tcy_0+1 in the vertical direction for the 9-7 wavelet filter. Each line 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 level shifted image tile component line of samples and produces as output one line of transform coefficients. In this example, it is assumed throughout this section that the image tile component has at least five rows.

J.2.1.1 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 line of samples of the level shifted image tile component and transfer this line of samples in the buffer *buf*.

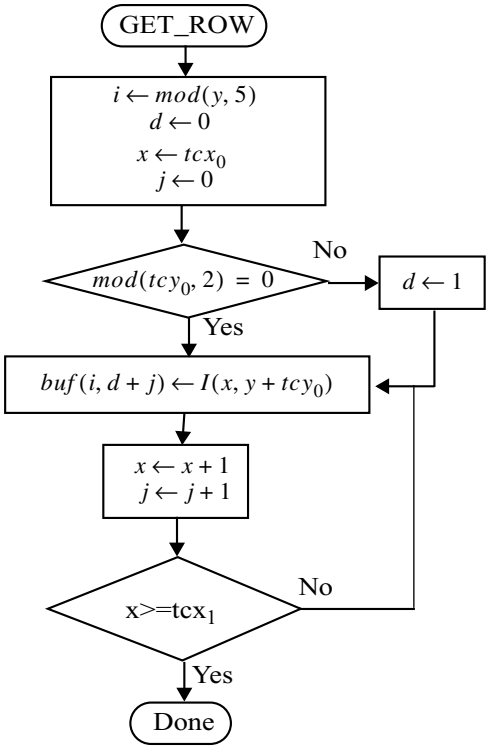


Figure J-5 — The GET_ROW procedure

J.2.2 The INIT procedure

As illustrated in Figure J-6, the INIT procedure reads five lines of samples of the level shifted image tile component and transfer these lines of samples in the buffer, *buf*.

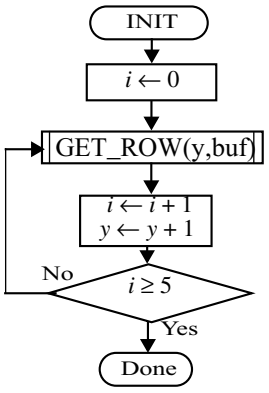


Figure J-6 — The INIT procedure

J.2.3 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 Figure of this section, the expression $buf(i) \leftarrow buf(i) + \alpha \cdot buf(i_2)$ is equivalent to $buf(i, j) \leftarrow buf(i, j) + \alpha \cdot buf(i_2, j)$ for tcx_1-tcx_0

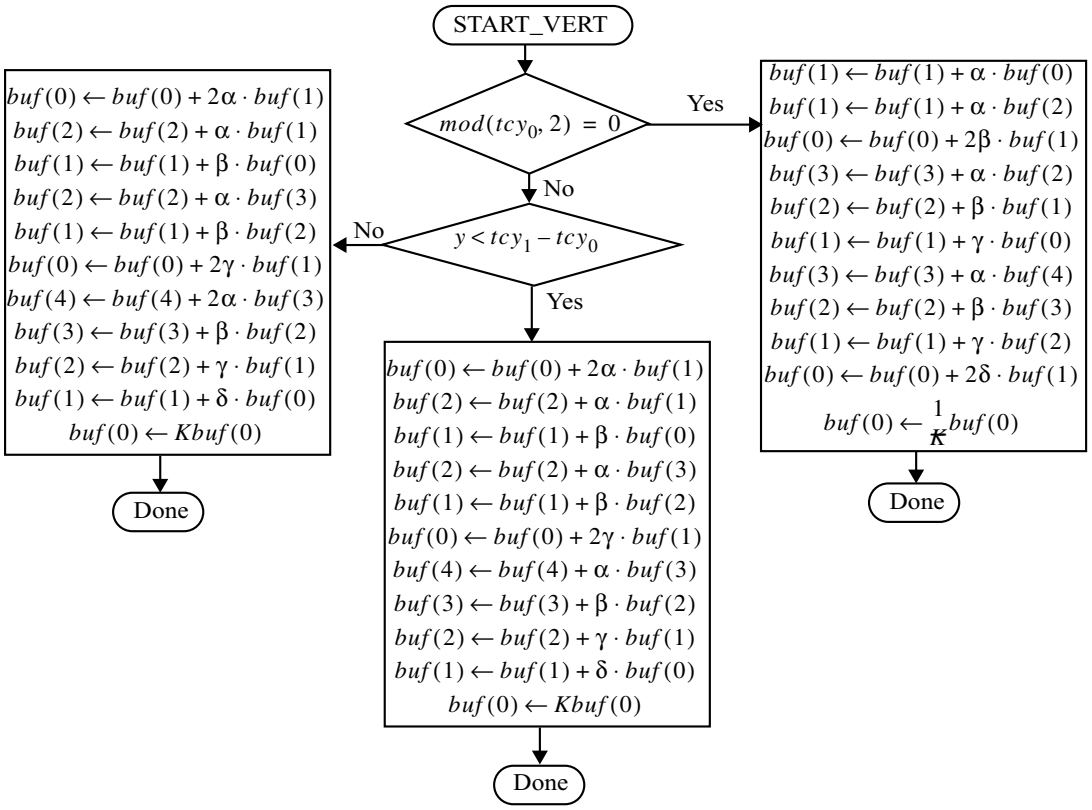


Figure J-7 — The START_VERT procedure

J.2.3.1 The RB_VERT_1 procedure

As illustrated in Figure J-8, the RB_VERT_1 procedure modifies the coefficient in buf(i,j).

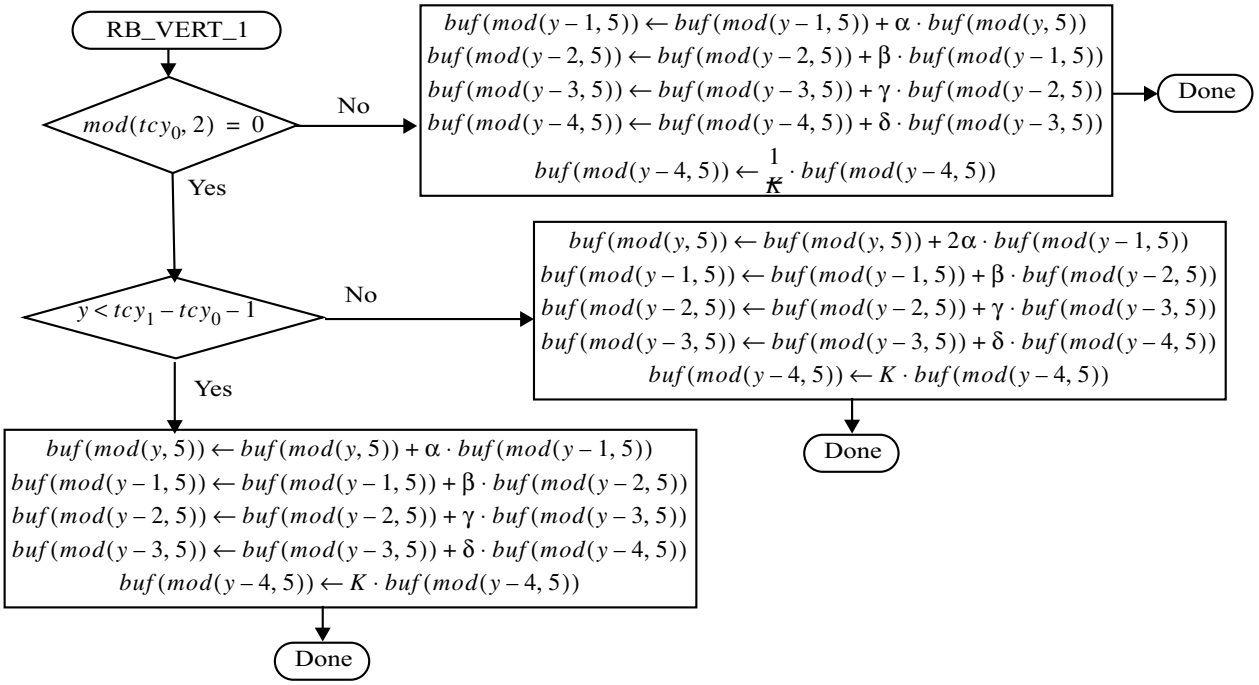


Figure J-8 — The RB_VERT_1 procedure

J.2.3.2 The RB_VERT_2 procedure

As illustrated in Figure J-9, the RB_VERT_2 procedure modifies the coefficient in buf(i,j).

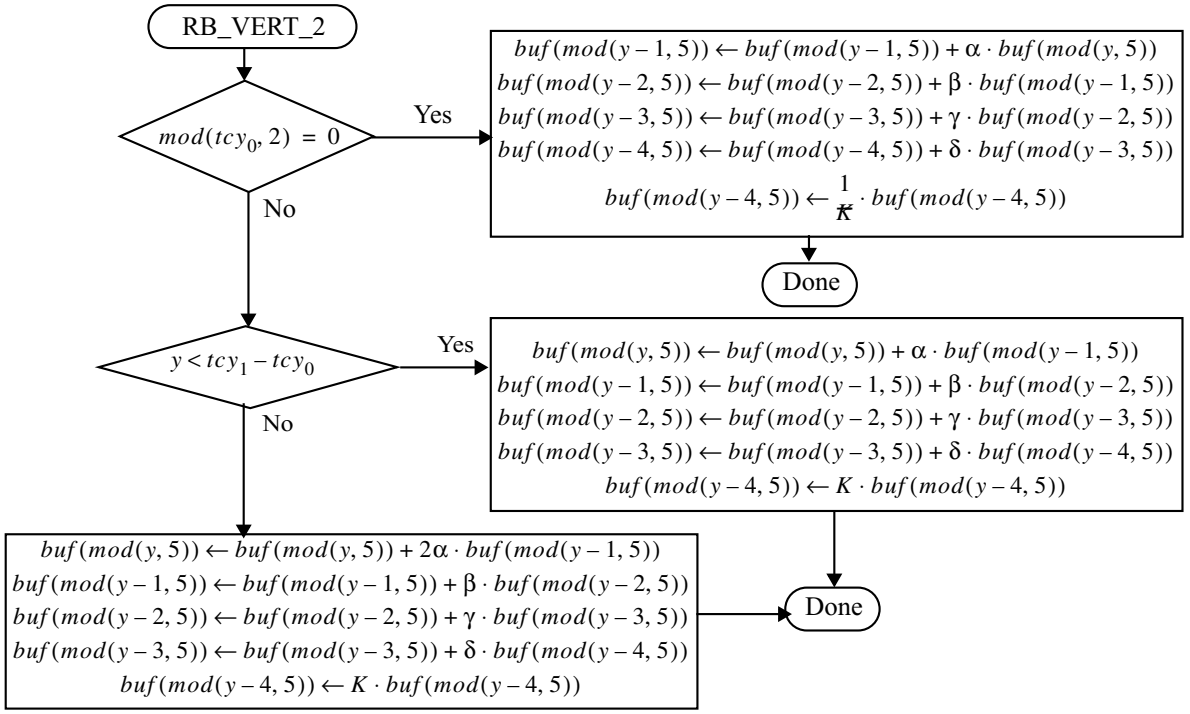


Figure J-9 — The RB_VERT_2 procedure

J.2.3.3 The END_1 procedure

The END_1 procedure is detailed in Figure J-10.

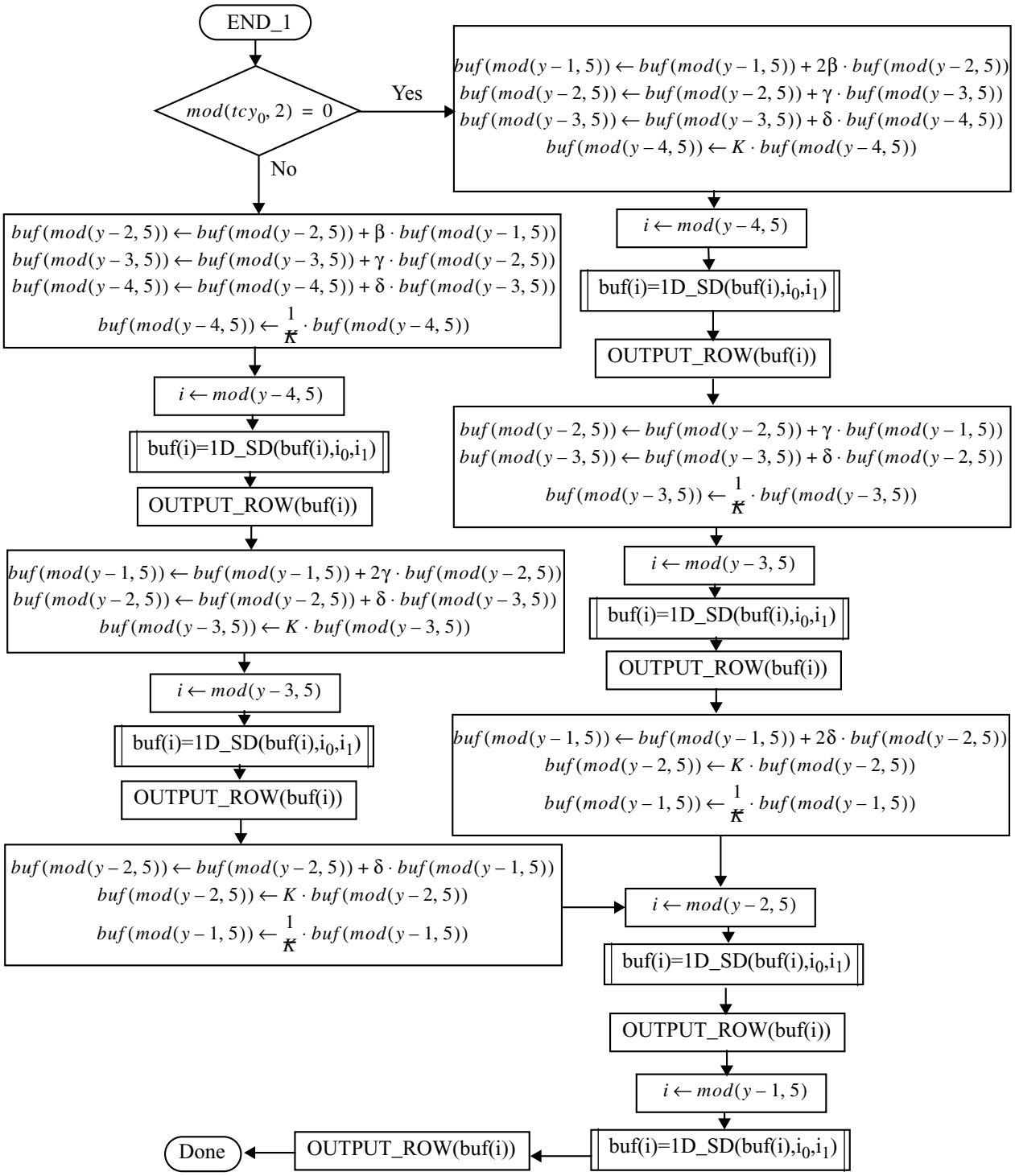


Figure J-10 — The END_1 procedure

J.2.3.4 The END_2 procedure

The END_2 procedure is detailed in Figure J-11

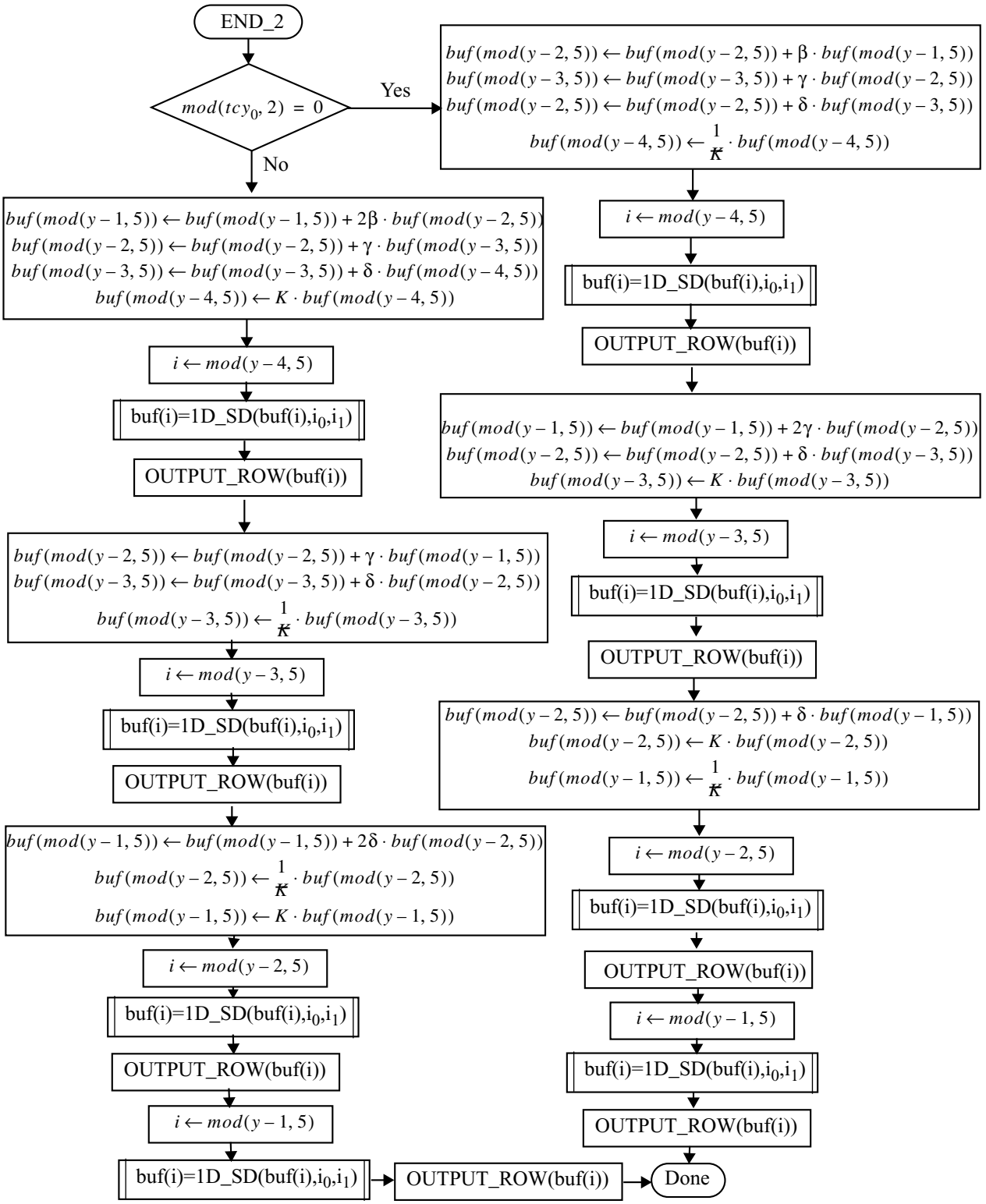


Figure J-11 — The END_2 procedure

J.2.4 OUTPUT_ROW procedure

This procedure returns a line *buf(i)* of transformed coefficients, which correspond either to the 1LL and 1HL sub-band or to the 1LH and 1HH sub-band. This line of transform coefficient can be either store in an external memory or processed immediately.

J.3 Scan-Based Coding

Some applications use scanning sensors that create images (possibly unconstrained in length) line by line and have limited amounts of memory available for processing purposes. These applications need a 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 system have been demonstrated [34][35]. The recommended procedure is outlined below.

Traditional JPEG2000 encoding requires all the wavelet coefficients to be buffered before quantization and coding. Alternatively, a scan-based approach can be used where the row-based wavelet transform (see Annex J.2) is followed by a scan-based rate allocation and coding procedure to ensure that wavelet coefficients are compressed soon after they have been generated. For this purpose, a limited memory buffer (the scan buffer) is introduced after the wavelet transform. The discrete 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 packet partition location, and corresponds to a small number of lines in image space. The scan buffer may contain one or more scan elements.

The rate control algorithm is applied to the 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.4 Error resilience

This section describes a method for decoding images, which have been coded using an error resilient syntax.

Many applications require the delivery of image data over different types of communication channels. Typical wireless communications channels give rise to random and burst bit errors. Internet communications are prone to loss due to traffic congestion. To improve the performance of transmitting compressed images over these error prone channels, error resilient bit stream syntax and tools are included in this specification.

The error resilience tools in this specification deal with channel errors using the following approaches: data partitioning and resynchronization, error detection and concealment, and Quality of Service (QoS) transmission based on priority. Error resilience tools are described in each category.

Table J-1 — Error resilience tools

Type of tool	Name	Reference
Entropy coding level	code-blocks termination of the arithmetic coder for each pass reset of contexts after each coding pass selective arithmetic coding bypass segmentation symbols	Annex D
Packet level	short packet format packet with resynchronization marker	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 to coder continue to decode coding passes after errors (see Annex 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 Annex D.6).

Short packets are achieved by moving the packet headers to the PPM or PPT marker segments (see Annex A.7.4 and Annex A.7.5). If there are errors, the packet headers in the PPM or PPT marker segments can still be associated with the correct packet by using the sequence number in the SOP.

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 Annex D.5.

A packet with a resynchronization marker SOP (see Annex 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 Annex B.9.

J.5 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 2.2.0 of the ICC Profile Format Specification. 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.5.1 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[1] defines this class of profile as Three-Color Matrix-Based Input Profiles (defined in Section 6.3.1.2 of the ICC profile format specification) and Monochrome Input Profiles (defined in Section 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:

$$\begin{aligned} linear_r &= redTRC[decompressed_r] \\ linear_g &= greenTRC[decompressed_g] \\ linear_b &= blueTRC[decompressed_b] \end{aligned} \qquad \text{J.1}$$

$$\begin{bmatrix} connection_x \\ connection_y \\ connection_z \end{bmatrix} = \begin{bmatrix} redColorant_x & greenColorant_x & blueColorant_x \\ redColorant_y & greenColorant_y & blueColorant_y \\ redColorant_z & greenColorant_z & blueColorant_z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} \qquad \text{J.2}$$

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.1, the three look-up tables are loaded from the Restricted ICC profile from the redTRCTag, greenTRCTag and blueTRCTag tags respectively, as defined in Sections 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 Section 6.5.25 of the profile specification. In Equation J.2, the rows of the matrix are loaded from the redColorantTag, greenColorantTag and blueColorantTag tags respectively, as defined in Sections 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 Section 6.5.2 of the profile specification.

The Monochrome Input Profile class can be described with the following equations:

$$connection = grayTRC[device] \qquad \text{J.3}$$

where device is the original decompressed pixel and connection is the achromatic channel of the profile connection space. In Equation J.3, the look-up table is loaded from the Restricted ICC profile from the grayTRCTag, as specified in Section 6.3.17. The data format of that tag is defined in Section 6.5.2 of the profile specification.

J.5.2 Converting images to sRGB[2]

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).

This transform[7]is used in conjunction with the Restricted ICC method to create resulting sRGB values from original source colour values. Where applicable, like transforms (1D look-up tables or matrices) may be combined to enhance processing performance. For this example, only the transform from the Profile Connection Space (XYZ_{PCS}) will be shown. It may later be combined with the transforms in Equation J.1 and Equation J.2

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 3x3 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 transform 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 transform which can be used. For this example transformation, the Bradford chromatic adaptation transform[3] (BFD) will be used. The Bradford transform has been shown to produce accurate results[4,5]and has been adopted as part of the CIE recommended colour appearance model[4] (CIECAM97s). The BFD transform typically includes a linear and a non-linear portion. In the case of this example transform, the non-linear portion of the Bradford transform has been left out to allow for simple 3x3 matrix processing. It has been shown that the Bradford transform's performance is still very good even with this omission[6].

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 transform looks as follows with the primary transformation denoted by a PT and the Bradford chromatic adaptation matrix denoted by a BFD:

$$\begin{bmatrix} slinear_r \\ slinear_g \\ slinear_b \end{bmatrix} = \begin{bmatrix} 3.2406_{PT} & -1.5372_{PT} & -0.4986_{PT} \\ -0.9689_{PT} & 1.8758_{PT} & 0.0415_{PT} \\ 0.0557_{PT} & -0.2040_{PT} & 1.0570_{PT} \end{bmatrix} \begin{bmatrix} 0.9554_{BFD} & -0.0231_{BFD} & 0.0633_{BFD} \\ -0.0284_{BFD} & 1.0100_{BFD} & 0.0211_{BFD} \\ 0.0123_{BFD} & -0.0205_{BFD} & 1.3305_{BFD} \end{bmatrix} \begin{bmatrix} connection_x \\ connection_y \\ connection_z \end{bmatrix} \qquad \text{J.4}$$

However, the matrices can be combined to form a single matrix as shown in the following equation:

$$\begin{bmatrix} slinear_r \\ slinear_g \\ slinear_b \end{bmatrix} = \begin{bmatrix} 3.1337 & -1.6173 & -0.4907 \\ -0.9785 & 1.9162 & 0.0334 \\ 0.0720 & -0.2290 & 1.4056 \end{bmatrix} \begin{bmatrix} connection_x \\ connection_y \\ connection_z \end{bmatrix} \qquad \text{J.5}$$

It is then necessary to transform the slinear rgb's to non-linear sRGB values. This is done through the following two equations:

If $slinear_r, slinear_g, slinear_b \leq 0.0031308$

$$\begin{aligned} sRGB_r &= 12.92 \times slinear_r \\ sRGB_g &= 12.92 \times slinear_g \\ sRGB_b &= 12.92 \times slinear_b \end{aligned} \quad \text{J.6}$$

If $slinear_r, slinear_g, slinear_b > 0.0031308$

$$\begin{aligned} sRGB_r &= 1.055 \times slinear_r^{(1.0/2.4)} - 0.055 \\ sRGB_g &= 1.055 \times slinear_g^{(1.0/2.4)} - 0.055 \\ sRGB_b &= 1.055 \times slinear_b^{(1.0/2.4)} - 0.055 \end{aligned} \quad \text{J.7}$$

where $sRGB_{rgb}$ is the pixels converted into the sRGB colourspace, and again $slinear_{rgb}$ is the pixel in the linear RGB form of sRGB.

Note that this processing can be optimized by combining the colourant matrix described in Equation J.2 with the XYZ to sRGB conversion matrix described in Equation J.5 as follows:

$$\begin{bmatrix} slinear_r \\ slinear_g \\ slinear_b \end{bmatrix} = \begin{bmatrix} 3.1337 & -1.6173 & -0.4907 \\ -0.9785 & 1.9162 & 0.0334 \\ 0.0720 & -0.2290 & 1.4056 \end{bmatrix} \begin{bmatrix} redColorant_x & greenColorant_x & blueColorant_x \\ redColorant_y & greenColorant_y & blueColorant_y \\ redColorant_z & greenColorant_z & blueColorant_z \end{bmatrix} \begin{bmatrix} linear_r \\ linear_g \\ linear_b \end{bmatrix} \quad \text{J.8}$$

This optimization reduces the colourspace processing from PCS XYZ to sRGB to the application of a 1D look-up table, a single 3x3 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[1,2,6,8].

J.5.3 Input and output ranges and quantization

The input code values to the look-up tables in Equation J.1 (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.1 shall be real linear intensity values nominally in the range (0.0, 1.0).

The input to the colourant matrix in Equation J.2 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 input and output of Equation J.4, and thus the input to Equations R.4 and J.7 are also real values in the range (0.0, 1.0).

The output of Equations R.4 and J.7 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.5.4 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 atom 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 atom 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.

J.6 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. By-passing the colour space transform, the ensemble of chips may then be subjected to lossless or lossy compression. This procedure has two advantages: all the ROIs in a given data set can be compressed in a single pass; and bit allocation can be optimized across the ensemble of ROIs rather than on a chip-by-chip basis.

J.7 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.

```
0000000 ff4f ff51 002a 0000 0000 0001 0000 0009
0000020 0000 0000 0000 0000 0000 0001 0000 0009
0000040 0000 0000 0000 0000 0001 0008 0101 ff5c
0000060 0007 4008 0909 0aff 5200 0b00 0100 0001
0000100 0404 0001 ff90 000a 0000 0000 001e 0001
0000120 ffda c7d4 0c01 8f0d c875 5da0 3e10 c00f
0000140 b176 ffd9
```

This bit stream contains the marker segments listed below.

Main header:

```
0000000 ff4f SOC marker
0000002 ff51 SIZ marker
0000004 002a   Lsiz SIZ marker length
0000006 0000   Rsiz
0000010 0000 0001 Xsiz
0000014 0000 0009 Ysiz
0000020 0000 0000 XOsiz
0000024 0000 0000 YOsiz
0000030 0000 0001 XTsiz
0000034 0000 0009 YTsiz
0000040 0000 0000 XTOsiz
```

```
0000044 0000 0000 YTOsiz
0000050 0001 Csiz
0000052 00   CSsiz
0000053 08   Ssiz
0000054 01   XRsiz
0000055 01   YRsiz
```

Thus the "image" is one component, with 8 bits/sample unsigned, 1 sample horizontally, and 9 samples vertically, an all samples are in a single tile.

```
0000056 ff5c QCD marker
0000060 0007   Lqcd QCD marker length
0000062 40     Sqcd
0000063 08 0909 0a 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}.

```
0000067 ff52 COD marker
0000071 000b   Lcod COD marker length
0000073 00     Scod
0000074 01     Decomposition level
0000075 00     Progression style
0000076 0001   Number of layers
0000100 04    Code block width exponent value
0000101 04    Code block height exponent value
0000102 00    Code block coding pass style
0000103 01    Transform
```

No packet partitions are used. There is one level of wavelet transform. Progression is layer-resolution-component-position, but there is only one layer. Code-blocks are 64x64 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 wavelet transform is used.

Tile-part header:

```
0000104 ff90 SOT marker
0000106 000a   Lsot SOT marker length
0000110 0000   Isot
0000112 0000 001e   Psot
0000116 00    TPsot
0000118 01   TNsot
```

This is tile number 0. The length is 30 bytes (octal 142 - 104). This is tile-part 0. There is only one tile-part for this tile.

```
0000120 ffda SOS marker
```

Coded Data (Packet headers and packet bodies)

```
0000122 c7d4 0c01 8f0d c875 5da0 3e10 c00f
0000140 b176
```

End of Image

```
0000142 ffd9 EOI marker
```

Because the image is 1x9, and there is one level of transform, (and the code-blocks, partitions, and tiles are two large to have an effect), there will be 5 low pass wavelet coefficients, and 4 horizontal low pass vertical high pass coefficients.

The LL sub-band is decoded as follows. The first item is the context label from Annex C (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.

```
17 C4(ZERO_RUN) Bit 1
```

No zero run occurred.

```
18 C5(UNIFORM) Bit 1
18 C5(UNIFORM) Bit 1
```

First nonzero coefficient is the 4th (numbered from 1).

```
9 C2(SIGN)       Bit 1
```

Negative.

```
3 C1(NEW_SIGNIFICANT)   Bit 0
```

Fifth coefficient is not significant.

```
3 C1(NEW_SIGNIFICANT)   Bit 1
```

Third coefficient is significant (first coefficient which is in the significance pass).

```
10 C2(SIGN)      Bit 0
```

Negative (XOR bit is 1).

```
3 C1(NEW_SIGNIFICANT)   Bit 1
```

Fifth coefficient is significant now.

```
 10 C2(SIGN)      Bit 0
```

Negative (XOR bit is 1).

```
15 C3(REFINE)   Bit 0
```

Next bit of 4th coefficient is 0.

```
 0 C1(NEW_SIGNIFICANT)   Bit 1
```

First coefficient is significant.

```
9 C2(SIGN)       Bit 1
```

Negative.

```
4 C1(NEW_SIGNIFICANT)   Bit 1
```

Second coefficient is significant.

```
10 C2(SIGN)      Bit 0
```

Negative.

Now all coefficients are in the refinement pass. Decoded bit is the next bit of the coefficient in order from 1st to fifth.

```
15 C3(REFINE)   Bit 1
15 C3(REFINE)   Bit 0
15 C3(REFINE)   Bit 1
16 C3(REFINE)   Bit 0
15 C3(REFINE)   Bit 0
```

Next bit-plane.

```
16 C3(REFINE)   Bit 0
16 C3(REFINE)   Bit 1
16 C3(REFINE)   Bit 1
16 C3(REFINE)   Bit 0
```

```
16 C3(REFINE)   Bit 0
```

Next bit-plane.

```
16 C3(REFINE)   Bit 1
16 C3(REFINE)   Bit 1
16 C3(REFINE)   Bit 1
16 C3(REFINE)   Bit 0
16 C3(REFINE)   Bit 1
```

Last bit-plane.

```
16 C3(REFINE)   Bit 0
16 C3(REFINE)   Bit 0
16 C3(REFINE)   Bit 0
16 C3(REFINE)   Bit 0
16 C3(REFINE)   Bit 1
```

Thus the decoded coefficients are:

```
-26, -22, -30, -32, -19
```

For the vertical high pass horizontal lowpass sub-band the following contexts and bits occur.

```
17 C4(ZERO_RUN) Bit 1
18 C5(UNIFORM) Bit 0
18 C5(UNIFORM) Bit 1
9 C2(SIGN)      Bit 0
3 C1(NEW_SIGNIFICANT)   Bit 0
0 C1(NEW_SIGNIFICANT)   Bit 0
3 C1(NEW_SIGNIFICANT)   Bit 0
3 C1(NEW_SIGNIFICANT)   Bit 0
14 C3(REFINE)   Bit 0
0 C1(NEW_SIGNIFICANT)   Bit 0
3 C1(NEW_SIGNIFICANT)   Bit 1
10 C2(SIGN)     Bit 0
3 C1(NEW_SIGNIFICANT)   Bit 1
10 C2(SIGN)     Bit 0
3 C1(NEW_SIGNIFICANT)   Bit 0
16 C3(REFINE)   Bit 1
```

The decoded vertical high pass horizontal low pass coefficients are:

```
1, 5, 1, 0
```

After the inverse 5,3 wavelet transform and level shifting, the component samples in decimal are:

```
 101,103,104,105,96,97,96,102,109
```

J.8 Visual Frequency Weighting

The human visual system plays an important role in the perceived image quality of compressed images. It is therefore desirable to allow system designers and users to take advantage of the current knowledge of visual perception, e.g., to utilize models of the visual system's varying sensitivity to spatial frequencies, as measured in the contrast sensitivity function (CSF). Since the CSF weight is determined by the visual frequency of the transform coefficient, there will be one CSF weight per sub-band in the wavelet transform. The design of the CSF weights is an encoder issue and depends on the specific viewing condition under which the decoded image is to be viewed. Please refer to [29][30] 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.

J.8.1 Fixed Visual Weighting

In fixed visual weighting, a set of CSF weights, $\{w_i\}$, is chosen according to the final viewing condition, where w_i is the weight for the ith sub-band. The set of CSF weights can be incorporated in one of the following two ways.

J.8.1.1 Modify Quantization Step Size

At the encoder, the quantization step size q_i of the transform coefficients of the ith sub-band is adjusted 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.8.1.2 Modify the embedded coding order

The quantization step sizes are not modified but the distortion weights fed into the R-D optimization are altered instead. This effectively controls the relative significance of including different numbers of bit-planes from the embedded bit stream of each code-block. The frequency-weighting table does not need to be transmitted explicitly. This approach is recommended since it produces similar results in Annex J.8.1.1 and is compatible with lossless compression. This approach affects only the compressor and it is compatible with all quantization strategies, including implicit quantization.

J.8.2 Visual progressive coding (VIP)

If the visual frequency weights are to be changed during the embedded coding process, it is very clumsy to change the coefficient values or quantization step sizes. Furthermore, the performance of the subsequent entropy coder may degrade due to the changing statistics of the binary representation. An elegant way to implement the visual progressive coding (VIP) is to change, on the fly, the order in which code-block sub-bit-planes should appear in the overall embedded bit stream based on the visual weights, instead of changing the coefficient values or quantization step sizes. In other words, the coding order rather than the coding content is affected by the visual weights.

A series of visual weighting sets for different bit rate ranges are denoted as follows:

$$\begin{aligned} &\text{Weighting set 0: } r(0), \text{ with } W(0) = \{w_0(0), w_1(0), \ldots, w_n(0)\}; \\ &\text{Weighting set 1: } r(1), \text{ with } W(1) = \{w_0(1), w_1(1), \ldots, w_n(1)\}; \\ &\qquad\qquad \ldots \\ &\text{Weighting set m: } r(m), \text{ with } W(m) = \{w_0(m), w_1(m), \ldots, w_n(m)\}, \end{aligned} \tag{J.9}$$

where r(j) represents a bit-rate at which the weighting factors are changed, $r(0) < r(1) < \ldots < r(m)$, 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, the progressively changing visual weights effectively control the embedding order of code-block sub-bit-planes on the fly.

J.8.3 Recommended frequency weighting tables

The following table 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 pixels (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.

The table does not include the weight for the lowest frequency sub-band, nLL, which is always 1. Levels 1, 2, …, 5 denote the sub-band levels in low to high frequency order. (HL, LH, HH) denotes the three frequency orientations within each sub-band.

Table J-2 — Recommended frequency weighting

level	Viewing distance (pixels) 1000			Viewing distance (pixels) 2000			Viewing distance (pixels) 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 color 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.

J.9 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 transform followed by sub-sampling of the chrominance (Cr, Cb) components. While this is an effective way to reduce the amount of data to encode for DCT-based compression algorithms (ITU-T Recommendation T.81 | ISO/IEC 10918-1:1994), it is not recommended for use in this Recommendation | International Standard.

The multi-resolution nature of the wavelet transform 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 beneficial in terms of image quality to retain a few of the wavelet coefficients in the 1HL, 1LH, 1HH sub-bands, 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.10 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.10.1 Introduction to key concepts for rate control

Divide each sub-band 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 $\{B_i\}_{i = 1, 2, \ldots}$ 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,and the distortion incurred when reconstructing from each of these truncated subsets is estimated and denoted by D_i^1, D_i^2, D^3,... The Mean Squared Error distortion metric is 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 Annex J.10.2.

J.10.2 Layered Bit-Stream Abstraction

An important aspect is the manner by which it 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 1. 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 (that is, the bit-stream contains twice as much granularity as that provided by previous verification models). 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 1 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

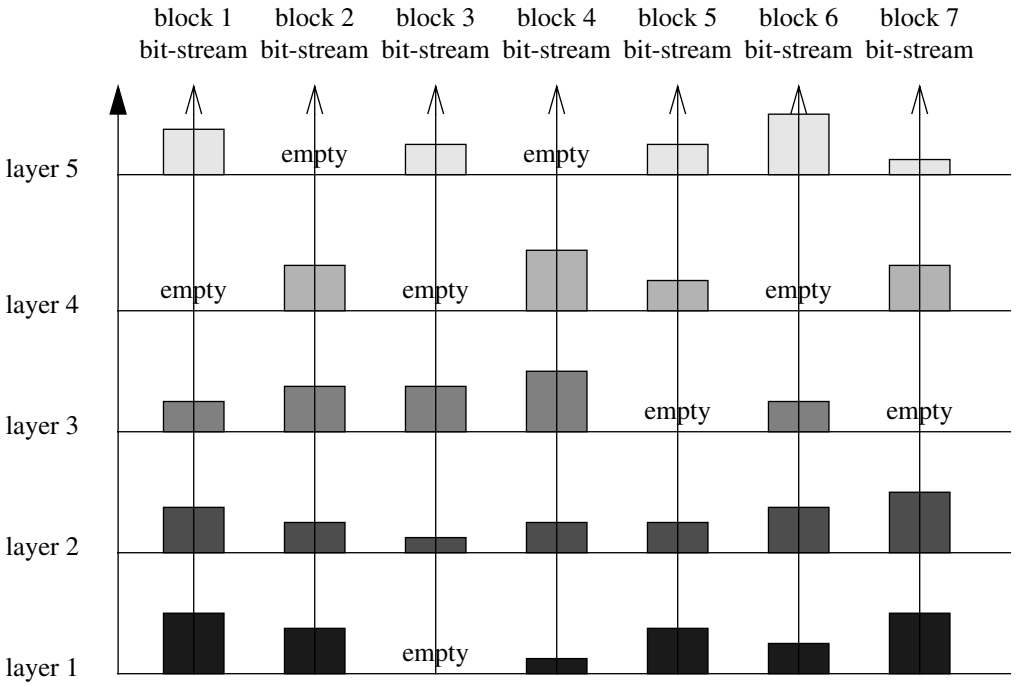


Figure J-12 — Illustration of code-block contributions to bit-stream layers

J.10.3 Rate-Distortion Optimization

The rate-distortion algorithm described here is justified strictly only provided the distortion measure adopted for the code-blocks is additive. That is, the distortion, D , in the final reconstructed image should satisfy

$$D = \sum_i D_i^{n_i}, \qquad \text{J.10}$$

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 transform is orthogonal. Additivity also holds if the quantization errors for individual sample values are uncorrelated, regardless of whether or not the transform is orthogonal. In practice, the transform 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

$$R = \sum_i R_i^{n_i} \qquad \text{J.11}$$

The need is to find the set of n_i values which minimizes D subject to the constraint $R \leq R_{max}$. The solution to this constrained optimization problem by the method of Lagrange multipliers is well known. Specifically, the problem is equivalent to minimizing

$$\sum_i (R_i^{n_i} - \lambda D_i^{n_i}) \qquad \text{J.12}$$

where the value of λ must be adjusted until the rate yielded by the truncation points which minimize Equation J.12 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.12 for some λ is guaranteed to be optimal in the sense the 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.12, 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 _, 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.12, it is a separate optimization problem for each individual code-block. Specifically, for each code-block, B_i, the truncation point, n_i ,need to be found which minimizes $(R_i^{n_i} + \lambda D_i^{n_i})$. 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 $\Delta D_i^k = D_i^{n_i} - D_i^k$

If $\Delta D_i^k / \Delta R_i^k > \lambda^{-1}$ 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 is simply to set 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 very much smaller than the bit-stream itself which is stored for the code-block.

J.10.4 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^n , 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 the previous chapter, which is exploited in all our experimental investigations.

J.10.4.1 Considerations for Non-Reversible Transforms

The rate-distortion optimization algorithm described in the previous chapter 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 compute 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 lookup 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

$$\mathfrak{v}_i^p[m,n] = 2^{-p} v_i[m,n] - 2\left\lfloor \frac{2^{-p} v_i[m,n]}{2} \right\rfloor \qquad \text{J.13}$$

Thus, $\mathfrak{v}_i^p[m,n]$ 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 $0 \le \mathfrak{v}_i^p[m,n] \le 2$. 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.13 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 $\mathfrak{v}_i^p[m,n] \ge 1$ and the reduction in distortion may be expressed as

$$2^{2p}\omega_i\Delta_i^2\left((\mathfrak{v}_i^p[m,n])^2 - \left(\mathfrak{v}_i^p[m,n] - \frac{3}{2}\right)^2\right) = 2^{2p}\omega_i\Delta_i^2 \cdot f_s(\mathfrak{v}_i^p[m,n]) \qquad \text{J.14}$$

provided the representation levels used during inverse quantization are midway between the quantization thresholds, which is the case in our implementation. Also, the reduction in distortion which may be attributed to magnitude refinement of a sample in bit-plane p may be expressed as

$$2^{2p}\omega_i\Delta_i^2\left((\mathfrak{v}_i^p[m,n] - 1)^2 - \left(\mathfrak{v}_i^p[m,n] - \frac{1}{2}\right)^2\right) = 2^{2p}\omega_i\Delta_i^2 \cdot f_m(\mathfrak{v}_i^p[m,n]) \qquad \text{J.15}$$

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, $\mathfrak{v}_i^p[m,n]$, has a binary representation of the form $v.\text{xxxxx}$, where v, the only bit before the binary point, is simply the value of magnitude bit p, i.e. $v_i^p[m,n]$. In the implementation, exactly 6 extra bits beyond the binary point are used to index a 7-bit lookup table for $f_m(.)$ and a 6-bit lookup table for $f_s(.)$ (recall that we must have $1 \le \mathfrak{v}_i^p[m,n] < 2$ when a sample first becomes significant). Each entry of these lookup tables holds a 16-bit fixed point representation of $2^{13} f_s(\mathfrak{v}_i^p[m,n])$ or $2^{13} f_m(\mathfrak{v}_i^p[m,n])$, 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.10.4.2 Considerations for Reversible Transforms

By and large the process for estimating distortion whilst encoding the coefficients produced by a reversible transform is no different to that for a non-reversible transform. There are, however, two subtle differences which must be pointed out here. Equation J.14 and Equation J.15 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 behavior for the quantizer most of the time, except for the least significant bit-plane in the reversible mode. In this case $\Delta_i = 1$ and there is 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, Equation J.14 and Equation J.15 should be modified to

$$\left(2^{2p}\omega_i\Delta_i^{\,2}(\mathfrak{v}_i^{\,p}[m,n])\right)^2 = 2^{2p}\omega_i\Delta_i^{\,2}\cdot f'_s(\mathfrak{v}_i^{\,p}[m,n]) \qquad \text{J.16}$$

and

$$\left(2^{2p}\omega_i\Delta_i^{\,2}(\mathfrak{v}_i^{\,p}[m,n]-1)\right)^2 = 2^{2p}\omega_i\Delta_i^{\,2}\cdot f'_m(\mathfrak{v}_i^{\,p}[m,n]) \qquad \text{J.17}$$

respectively.

Annex K

Bibliography

K.1 General

[1] C. Christopoulos, T. Ebrahimi, A. Skodras, "The upcoming JPEG2000 standard," *Invited Tutorial to the 11th Portuguese Conference on Pattern Recognition (RECPAD 2000)*, Porto, Portugal, 11-12 May 2000

[2] T. Ebrahimi, C, Christopoulos, "JPEG 2000 standard: Still image compression scheme of 21st century," *JPEG2000 Tutorial to be presented in to the European Signal processing Conference (EUSIPCO 2000)*, Tampere, Finland, 5-8 September 2000.

[3] M. W. Marcellin, M. Gormish, A. Bilgin, M. Boliek, "An Overview of JPEG 2000," *Proc. of Data Compression Conference*, Snowbird, Utah, March 2000.

[4] A. Zandi, J. D. Allen, E. L. Scwhartz, M. Boliek, "CREW: Compression with reversible embedded wavelets," *Proc. of Data Compression Conference*, Snowbird, UT, pp. 212-21, March 1995.

K.2 Wavelet transform

[5] M.D. Adams and F. Kossentini, "Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis," *IEEE Trans. on Image Processing*, 2000.

[6] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using the wavelet transform," *IEEE Trans. Image Proc. 1*, pp. 205-220, April 1992.

[7] C. M. Brislawn, "Classification of nonexpansive symmetric extension transforms for multirate filter banks," *Appl. Comput. Harmonic. Analysis*, vol. 3, pp. 337-57, 1996.

[8] A.R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332-369, July 1998.

[9] C. Chrysafis, A. Ortega, "An algorithm for low memory wavelet image compression," *Proc. IEEE Int. Conference on Image Processing (ICIP)*, 24-28 October 1999, Kobe, Japan.

[10] C.K. Chui, *An Introduction to Wavelets*, Academic Press, Boston, 1992.

[11] C.K. Chui, *Wavelets: A Mathematical Tool for Signal Analysis*, SIAM Publ., Philadelphia, 1997.

[12] I. Daubechies, *Ten Lectures on Wavelets*, SIAM Publ., Philadelphia, 1992.

[13] D. Le Gall and A. Tabatabai, "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques," IEEE International Conference on Acoustics, Speech and Signal Processing, New York, NY, pp. 761–765, 1988.

[14] S. Mallat, *A Wavelet Tour of Signal Processing*, Second Edition, Academic Press, San Diego, 1999.

[15] W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmonic. Analysis,* vol. 3, no. 2, pp 186-200, 1996.

[16] W. Sweldens, "The lifting scheme: construction of second generation wavelets," *SIAM J.Math. Anal*., vol. 29, no. 2, pp 511-546, 1997.

K.3 Quantization and Entropy coding

[17] E. Ordentlich, M. J. Weinberger, G. Seroussi, "A low-complexity modelling approach for embedded coding of wavelet coefficients," *Proc. of Data Compression Conference,* Snowbird, Utah, pp. 408-417, March 29-April 1, 1998.

[18] E. Ordentlich, D. Taubman, M.J. Weinberger, G. Seroussi, and M. Marcellin, "Memory Efficient Scalable Line-based Image Coding," *Proc. of Data Compression Conference*, Snowbird, Utah, pp. 218-227, March 29-31, 1999.

[19] D. Taubman, A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Trans. Image Processing*, vol. 3, pp. 572-88, September 1994.

[20] D. Taubman, "High performance scalable image compression with EBCOT", *Proc. IEEE Int. Conference on Image Processing (ICIP)*, 24-28 October 1999, Kobe, Japan.

[21] D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Trans. on Image Processing*, June, 2000.

[22] J.W. Woods, J. Naveen, "A filter based bit allocation scheme for subband compression of HDJV," *IEEE Trans. on Image Processing*, July 1992.

K.4 Region of Interest coding

[23] E. Atsumi, N. Farvardin, "Lossy/lossless region-of-interest image coding based on set partitioning in hierarchical trees," Proc. IEEE International Conference on Image Processing (ICIP-98), pp. 87-91, 4-7 October 1998, Chicago, Illinois

[24] Nister D. and Christopoulos C., "Lossless Region of Interest with a naturally progressive still image coding algorithm," Proceedings of IEEE International Conference on Image Processing (ICIP 98), pp. 856-860, 4-7 October 1998, Chicago, Illinois,

[25] D. Nister, C. Christopoulos, "Lossless region of interest with embedded wavelet image coding," Signal Processing, Vol. 78, No. 1, pp. 1-17, October 1999.

[26] D. Santa Cruz, M. Larsson, J. Askelof, T. Ebrahimi, C. Christopoulos, "Region of Interest coding in JPEG2000 for interactive client/server applications," IEEE International Workshop on Multimedia Signal Processing, Copenhagen, Denmark, 13-15 September 1999.

K.5 Visual frequency weighting

[27] M. Albanesi and S. Bertoluzza, "Human vision model and wavelets for high-quality image compression," Proc. of 5th Int. Conference in Image Processing and its Applications, Edinburgh, UK, no. 410, pp. 311-315, 4-6 July 1995.

[28] M. Eckert, "Lossy compression using wavelets, block DCT, and lapped orthogonal transforms optimised with a perceptual model," SPIE vol. 3031, pp. 339-350, 1997.

[29] T. O'Rourke and R. Stevenson, "Human visual system based wavelet decomposition for image compression," J. VCIP V. 6, pp. 109-121, 1995.

[30] Watson, G. Yang, J. Solomon, and J. Villasenor, "Visibility of wavelet quantization noise," IEEE Trans. on Image Proc., vol. 6, pp. 1164-1175, 1997.

K.6 Error resilience

[31] H. Man, F. Kossentini and M. Smith, "A Family of Efficient and Channel Error Resilient Wavelet/Subband Image Coders," Special issue of the IEEE Transactions on Circuits and Systems for Video Technology on Interactive Multimedia, 9(2), February 1999.

[32] J. Liang and R. Talluri, "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.

[33] I. Moccagata, S. Sodagar, J. Liang and H. Chen, "Error Resilient Coding in JPEG-2000 and MPEG-4," IEEE Journal of Selected Areas in Communications (JSAC), April 2000.

K.7 Scan-based coding

[34] C. Lambert-Nebout, C. Latry, G. Moury, M. Antonini, M. Barlaud, C.Parisot "On-Board Optical image compression for future high resolution space remote sensing systems," Proc. SPIE, **Number? Month?** 2000.

[35] Thomas J. Flohr, Michael W.Marcellin, Janet C. Rountree, "Scan-Based Processing with JPEG 2000," Proc. SPIE, **Number? Month?** 2000.

K.8 ICC colour

[36] International Color Consortium (ICC), "ICC Profile Format Specification 1:1998-09," (1998).

[37] IEC TC100/61966-2.1,"Colour Management - Default RGB colour space -sRGB," (1999).

[38] K.M. Lam, "Metamerism and Colour Constancy," *University of Bradford* (1985)

[39] M. Ronnier Luo, Mei-Chun Lo and Wen-Guey Kuo, "The LLab(l:c) Colour Model," *Color Res. Appl.,* **21,** 412-429 (1996).

[40] M. R. Luo and R.W.G. Hunt, "A Chromatic Adaptation Transform and a Colour Constancy Index," *Color Res. Appl.,* **23,** 154-158 (1997).

[41] M.R. Luo and R.W.G. Hunt, "The Structure of the CIE 1997 Colour Appearance Model (CIECAM97s)," *Color Res. Appl.,* **23,** (1997).

[42] M. Nielsen and M. Stokes, "The Creation of the sRGB ICC Profile," *Proceedings IS&T/SID 6th Color Imaging Conference,* 253-257 (1998).

[43] E. Giorgianni and T. Madden, "Digital Color Management: Encoding Solution," Addision-Wesley, 1998.

Index

B
bit-stream 2
C
CME 16
COC 15
COD 15
codestream 2
coding blocks 93
context label 94
context vector 93
D
Decoder 8
Delimiting markers 21
E
Encoder 8
End of image 25, 50
EOI 15, 25, 50
F
file format 3
Fixed information marker 26
Functional markers 29
H
Headers 13
I
IEM 16
IET 16
Image and tile size 26
Image tiles 54
M
markers 13
Q
QCC 16
QCD 16
quantization 107
R
ROI 16
S
significance state 93
SIZ 15, 26
SOC 15, 21
SOS 15, 24
SOT 15, 22
Start of codestream 21
Start of scan 24
Start of tile 22, 49
T
Tile packets 67
TLM 16

Annex L

Patent Statement

There is the possibility that, for some of the processes specified in this Recommendation | International Standard, conformance or compliance may require use of an invention covered by patent rights.

By publication of this Recommendation | International Standard, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. Information regarding such patents can be obtained from the any organizations. The table summarizes the formal patent and intellectual property rights statements that have been received.

Table L-1 — Received intellectual property rights statements

Number	Company
1	Algo Vision
2	Canon Incorporated
3	Digital Accelerator Corporation
4	Digital Imaging Group (DIG)
5	Ericsson Corporation
6	Hewlett Packard Company
7	International Business Machines, Inc.
8	LizardTech, Incorporated
9	LuraTech
10	MITRE Incorporated
11	Mitsubishi Electric Corporation
12	Motorola Corporation
13	PrimaComp Incorporated
14	Rensselaer Polytechnic Institute (RPI)
15	Ricoh Company, Limited
16	SAIC
17	Sarnoff Corporation
18	Sharp Corporation
19	Sony Corporation
20	TeraLogic Incorporated

Table L-1 — Received intellectual property rights statements

Number	Company
21	University of Arizona
22	Washington State University