INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

T.87

(06/98)

SERIES T: TERMINALS FOR TELEMATIC SERVICES

Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline

ITU-T Recommendation T.87

(Previously CCITT Recommendation)

ITU-T T-SERIES RECOMMENDATIONS

TERMINALS FOR TELEMATIC SERVICES

For further details, please refer to ITU-T List of Recommendations.

INTERNATIONAL STANDARD 14495-1

ITU-T RECOMMENDATION T.87

INFORMATION TECHNOLOGY – LOSSLESS AND NEAR-LOSSLESS COMPRESSION OF CONTINUOUS-TONE STILL IMAGES – BASELINE

Summary

This ITU-T Recommendation | ISO/IEC International Standard defines a set of lossless (bit-preserving) and nearly lossless (where the error for each reconstructed sample is bounded by a pre-defined value) compression methods for coding continuous-tone, gray-scale, or colour digital still images.

This ITU-T Recommendation | ISO/IEC International Standard:

- specifies a process for converting source image data to compressed image data;
- specifies processes for converting compressed image data to reconstructed image data;
- specifies coded representations for compressed image data;
- provides guidance on how to implement these processes in practice.

Source

The ITU-T Recommendation T.87 was approved on the 18th of June 1998. The identical text is also published as ISO/IEC International Standard 14495-1.

FOREWORD

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

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1.

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

NOTE

In this Recommendation the term *recognized operating agency (ROA)* includes any individual, company, corporation or governmental organization that operates a public correspondence service. The terms *Administration*, *ROA* and *public correspondence* are defined in the *Constitution of the ITU (Geneva, 1992).*

CONTENTS

Included diskette:

– JPEG-LS reference implementation

– JPEG-LS conformance testing image set

– Auxiliary programs and examples.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – LOSSLESS AND NEAR-LOSSLESS COMPRESSION OF CONTINUOUS-TONE STILL IMAGES – BASELINE

1 Scope

This Recommendation | International Standard defines a set of lossless (bit-preserving) and nearly lossless (where the error for each reconstructed sample is bounded by a pre-defined value) compression methods for coding continuous-tone, gray-scale, or colour digital still images.

This Recommendation | International Standard

- specifies a process for converting source image data to compressed image data;
- specifies processes for converting compressed image data to reconstructed image data;
- specifies coded representations for compressed image data;
- provides guidance on how to implement these processes in practice.

2 Normative references

The following Recommendations and International Standards contain provisions which, through references 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 the currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- CCITT Recommendation T.81 (1992) | ISO/IEC 10918-1:1994, *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines.*
- ITU-T Recommendation T.83 (1994) | ISO/IEC 10918-2:1995, *Information technology – Digital compression and coding of continuous-tone still images: Compliance testing.*
- ITU-T Recommendation T.84 (1996)| ISO/IEC 10918-3:1997, *Information technology – Digital compression and coding of continuous-tone still images: Extensions.*
- ITU-T Recommendation T.84/Amd.1[1] | ISO/IEC 10918-3/Amd.1[1], *Information technology – Digital compression and coding of continuous-tone still images: Extensions – Amendment 1.*

2.2 Additional references

- ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange.*
- ISO 5807:1985, *Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.*
- ISO/IEC 9899:1990, *Programming languages – C.*

[1] Currently at the stage of draft.

3 Definitions, abbreviations, symbols and conventions

3.1 Definitions

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

3.1.1 ⌊ i ⌋, floor: Indicates the largest integer not exceeding *i*.

3.1.2 ⌈ i ⌉, ceiling: Indicates the smallest integer not exceeded by *i*.

3.1.3 abs(i): The absolute value of *i* : $-i$ if $i < 0$, *i* otherwise.

3.1.4 abbreviated format: A representation of compressed image data which is missing some or all of the mapping table specifications required for decoding, or a representation of mapping tables without frame headers, scan headers, and coded image data.

3.1.5 application environment: The standards for data representation, communication, or storage, which have been established for a particular application.

3.1.6 bias: Deviation from zero of accumulated prediction errors.

3.1.7 bit stream: Partially encoded or decoded sequence of bits.

3.1.8 causal template: A set of fixed relative positions of samples (with respect to the current sample being coded) which have been previously coded according to a pre-specified scan sequence.

3.1.9 coded image data segment: The coded representation of one restart interval.

3.1.10 coding: Encoding or decoding.

3.1.11 coding parameters: Integers used to specify the encoding process.

3.1.12 (coding) process: A general term for referring to an encoding process, a decoding process, or both.

3.1.13 colour image: A continuous-tone image that has more than one component.

3.1.14 columns: Samples per line in a component.

3.1.15 component: One of the two-dimensional arrays which comprise an image.

3.1.16 compressed data: Either compressed image data or parameter specification data or both.

3.1.17 compressed image (data): A coded representation of an image, as specified in this Recommendation | International Standard.

3.1.18 compression: Reduction in the number of bits used to represent source image data.

3.1.19 context: Function of samples in the causal template used to determine the coding of the present sample.

3.1.20 context modelling: Procedure estimating a probability distribution of prediction error from the context.

3.1.21 continuous-tone image: An image whose components have more than one bit per sample.

3.1.22 decoder: An embodiment of a decoding process and a sample transformation process.

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

3.1.24 encoder: An embodiment of an encoding process.

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

3.1.26 frame: A group of one or more scans through the data of one or more of the components in an image.

3.1.27 frame header: A marker segment that contains a start-of-frame marker and associated frame parameters that are coded at the beginning of a frame.

3.1.28 Golomb coding: A special case of Huffman coding matched to geometric distributions.

3.1.29 (local) gradient: Either a vector of differences between values of samples in the causal template, or each difference separately.

3.1.30 gray-scale image: A continuous-tone image that contains only one component.

3.1.31 horizontal sampling factor: The relative number of horizontal samples of a particular component with respect to the number of horizontal samples in the other components.

3.1.32 Huffman encoding: A prefix coding procedure which assigns a variable length code to each input symbol, so that the total code length is minimised.

3.1.33 image: A set of two-dimensional arrays of integer data.

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

3.1.35 interchange format: The representation of compressed image data for exchange between application environments.

3.1.36 interleaved: The descriptive term applied to the repetitive multiplexing of groups of data from each component in a scan in a specified order.

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

3.1.38 JPEG-LS preset coding parameters: Coding parameters for which a normative set of default values is specified.

3.1.39 JPEG-LS preset parameters: A coding parameter or a mapping table specified in an LSE marker segment.

3.1.40 line interleaved: The mode of operation in which the interleaved entities are lines.

3.1.41 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.

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

3.1.43 lossy: A descriptive term for encoding and decoding processes which are not lossless.

3.1.44 mapping table: A table used in a sampling mapping procedure.

3.1.45 marker: A two-byte code in which the first byte is hexadecimal FF (X'FF') and the second byte is a value between 1 and hexadecimal FE (X'FE')

3.1.46 marker segment: A marker and associated set of parameters.

3.1.47 max(i,j): The largest of i or j : i if $i > j$, j otherwise.

3.1.48 min(i,j): The smallest of i or j : i if $i < j$, j otherwise.

3.1.49 minimum coded unit: The smallest group of samples that is coded.

3.1.50 near-lossless: A description term for lossy encoding and decoding processes and procedures in which the output of the decoding process is such that each reconstructed image sample differs from the corresponding one in the input to the encoding process by not more than a pre-specified value.

3.1.51 near-lossless coding: The mode of operation which refers to any one of the encoding process, decoding process, or both, defined in this Recommendation | International Standard in which some of the procedures are near-lossless.

3.1.52 non-interleaved: The descriptive term applied to the data processing sequence when the scan has only one component.

3.1.53 parameter specification data: The coded representation of the parameters used in the encoder and decoder.

3.1.54 point transform: Scaling of a sample.

3.1.55 precision: Number of bits allocated to a particular sample.

3.1.56 predicted sample value: The output from the predictor.

3.1.57 prediction correction: The procedure that compensates for systematic biases in prediction.

3.1.58 prediction error: Difference between the current sample and the predicted sample value.

3.1.59 predictor: The procedure that computes a predicted sample value from previously reconstructed samples.

3.1.60 procedure: A set of steps which accomplishes one of the tasks which comprise an encoding or decoding process.

3.1.61 reconstructed image (data): An image which is the output of a decoding process or a sample transformation process.

3.1.62 reconstructed sample: The sample value reconstructed by the decoder. This equals the original sample value in lossless coding, or differs from the original sample value by at most **NEAR** in magnitude in near-lossless coding.

3.1.63 regular mode: Mode of operation when coding samples while not in the run mode.

3.1.64 restart interval: The integer number of MCUs processed as an independent sequence within a scan.

3.1.65 restart marker: The marker that separates two restart intervals in a scan.

3.1.66 run: A sequence of consecutive samples whose values are identical for lossless coding, or are within the limits required for near-lossless coding, and which is contained in the current image line.

3.1.67 run length: Number of samples in a run.

3.1.68 run mode: Mode of operation while coding runs.

3.1.69 run interruption sample: The sample following the last sample in a run when the run terminates before the end of line.

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

3.1.71 sample interleaved: The mode of operation in which the interleaved entities are samples.

3.1.72 sample mapping procedure: A procedure that maps each sample value output by a decoding process to a reconstructed sample value by means of mapping tables.

3.1.73 sample transformation process: A sample mapping procedure followed by an inverse point transform.

3.1.74 sample value: A non-negative integer indicating the image information in an image sample.

3.1.75 scan: A single pass through the data for one or more of the components in the image.

3.1.76 scan header: A marker segment that contains a start-of-scan marker and associated scan parameters that are coded at the beginning of a scan.

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

3.1.78 unary code: The unary code of a non-negative integer number *n* is composed of *n* zero bits followed by a one bit.

3.1.79 vertical sampling factor: The relative number of vertical samples of a particular component with respect to the number of vertical samples in the other components in the frame.

3.2 Abbreviations

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

BPS Bits Per Sample

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: CCITT Rec. T.81 | ISO/IEC 10918-1, ITU-T Rec. T.83 | ISO/IEC 10918-2, and ITU-T Rec. T.84 | ISO/IEC 10918-3.

LSB Least Significant Bit

MCU Minimum Coded Unit

MSB Most Significant Bit

PGM Portable Grey Map

PPM Portable Pix Map

SPIFF Still Picture Interchange File Format

3.3 Symbols

The following symbols used in this Recommendation | International Standard are listed below. A convention is used that parameters which are fixed in value during the encoding of a scan are indicated in **boldface** capital letters, and variables which change in value during the encoding of a scan are indicated in *italicised* letters.

a, b, c, d	positions of samples in the causal template
A[0..366]	367 counters storing accumulated prediction error magnitudes
Ah	ignored field in scan header
Al	successive approximation bit position, low
AppendToBitStream()	a function in the C programming language
$\mathbf{APP_n}$	marker reserved for application segments
B[0..364]	365 counters for storing bias values
BASIC_T1, BASIC_T2, BASIC_T3	basic default threshold values
bpp	number of bits needed to represent **MAXVAL** (not less than 2)
C	SPIFF parameter
C[0..364]	365 counters storing prediction correction values
$\mathbf{C_i}$	component identifier
CLAMP	out-of-range value clamping function
COM	comment marker
ComputeRx()	a function in the C programming language
D1, D2, D3	local gradients
DNL	define-number-of-lines marker
DRI	define restart interval marker
EMErrval	*Errval* mapped to non-negative integers in run interruption mode
ENTRIES	number of yet unspecified mapping table entries
EOI	end-of-image marker
EOLine	end of line indicator, used in run mode
Errval	prediction error (quantized or unquantized, before and after modulo reduction)
g	the order of a Golomb code
GetNextSample()	a function in the C programming language
G(*k*)	Golomb code function
glimit	number of bits to which the length of a Golomb code word is limited
$\mathbf{H_i}$	horizontal sampling factor for the *i*th component
$\mathbf{H_{max}}$	largest horizontal sampling factor
i, j, n	integers
i..j	the set of integers between *i* and *j*, including *i* and *j*.
ILV	indication of the interleave mode used for the scan
Ix	the value of the current sample in the input image
J[0..31]	32 variables indicating order of run-length codes
JPG	marker reserved for JPEG extensions
$\mathbf{JPG_n}$	marker reserved for JPEG extensions
k	Golomb coding variable for regular mode
LG(*k, glimit*)	limited length Golomb code function
LIMIT	the value of *glimit* for a sample encoded in regular mode
Ll	JPEG-LS preset parameters marker segment length specifier
LSE	JPEG-LS preset parameters marker

m	modulo counter for restart marker
MErrval	*Errval* mapped to non-negative integers in regular mode
map	auxiliary variable for error mapping at run interruption
MAXTAB, MAXTABX	maximum index to a mapping table
MAXVAL	maximum possible image sample value over all components of a scan
MAX_C	maximum allowed value of *C*[0..364], equal to 127
MIN_C	minimum allowed value of *C*[0..364], equal to –128
ModRange()	a function in the C programming language
N[0..366]	367 counters for frequency of occurrence of each context
Nb	number of samples in an MCU
Nf	number of components in a frame
Nn[365..366]	2 counters for negative prediction error for run interruption
NEAR	difference bound for near-lossless coding
Ns	number of components in a scan
$\wp$	probability distribution
P	sample precision
Pt	point transform parameter
Px	predicted value for the current sample
Q	context, determined from *Q1*, *Q2*, *Q3*
Q1, Q2, Q3	region numbers to quantize local gradients
qbpp	number of bits needed to represent a mapped error value
Qi	one of the three quantized region numbers *Q1*, *Q2*, *Q3*
Quantize()	a function in the C programming language
Ra, Rb, Rc, Rd	reconstructed values of samples in the causal template
RANGE	range of prediction error representation
RegularModeProcessing	a label in the C programming language
RESET	threshold value at which *A*, *B*, and *N* are halved
RItype	index for run interruption coding
rk	*rg* is the *rk*-th power of 2
rg	Golomb code order for run mode, a power of 2
$\mathbf{RST_m}$	restart marker number *m*
RunModeProcessing	a label in the C programming language
RUNcnt	repetitive sample count for run mode
RUNindex	index for run mode order
RUNval	repetitive reconstructed sample value in a run
Rx	reconstructed value of the current sample
SIGN	auxiliary variable used to hold the sign of a context
$\mathbf{SOF_{55}}$	JPEG-LS frame marker
SOI	start-of-image marker
SOS	start-of-scan marker
TABLE[0..**MAXTAB**]	mapping table
T1, T2, T3	thresholds for local gradients
$\mathbf{Td_iTa_i}$	replaced field in scan header
TEMP	auxiliary variable used in the calculation of the Golomb variable in run interruption coding

TID	mapping table identification number
Tm_i	mapping table selector for the *i*th component
Tq_i	field in frame header (not used in the procedures of this Recommendation | International Standard).
VERS	SPIFF parameter
V_i	vertical sampling factor for the *i*th component
V_{max}	largest vertical sampling factor
Wt	width of table entries, in bytes
Wxy	number of bytes used to represent **Y** and **X**
x	current sample position
x_i	number of samples per line in the *i*th component
X	number of samples per line in the component with largest horizontal dimension
Xe	number of samples per line in the component with largest horizontal dimension, specified in an LSE marker segment
X'values'	values within the quotes are hexadecimal
Y	number of lines in the component with the largest vertical dimension
Ye	number of lines in the component with the largest vertical dimension, specified in an LSE marker segment
y_i	number of lines in the *i*th component

4 General description

4.1 Purpose

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

a) **Encoder**: An embodiment of an *encoding process*. An encoder takes as input *source image data* and *parameter specifications* and, by means of a specified set of *procedures*, generates as output *compressed image data*. Encoding procedures are specified in Annexes A, B, and D.

b) **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 *reconstructed image data*. Decoding procedures are described in Annex F.

c) **Interchange format**: A compressed image data representation which includes all parameter specifications used in the encoding process. The interchange format is for exchange between application environments. The interchange format is specified in Annex C.

4.2 Coding principles

The main procedures for the lossless (and near-lossless) encoding process specified in this Recommendation | International Standard are shown in Figure 1. Procedures in Figure 1 are presented in this clause in the order the encoding process is carried out (see Annex A).

In this Recommendation | International Standard, a source image is input to the encoder sample after sample in a pre-defined scan pattern, and lossless image compression is formulated as an inductive inference problem as follows. When coding the current sample, after having scanned past data, inferences can be made on the value of this sample by assigning a conditional probability $\wp$ for the value of the current image sample, conditioned on previously received samples. This inference method is called modelling. The minimum average code length contribution of the current sample is $-\log_2(\wp)$. For near-lossless image compression this principle is modified to use reconstructed values of the preceding samples (instead of the original values) as conditioning data. During the encoding process, shorter codes are assigned to more probable events. The decoder can reconstruct the conditional probability used to encode the current samples, since it depends only on already decoded data.

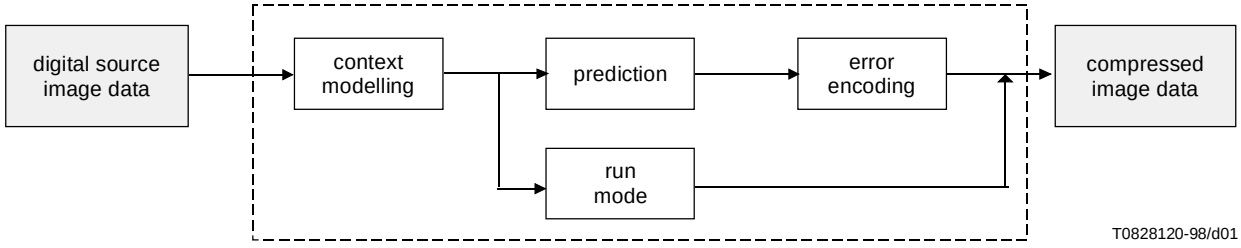


Figure 1 – Simplified encoder diagram

4.3 Source image

Source images to which the encoding process specified in this Recommendation | International Standard can be applied are defined in this clause.

4.3.1 Dimensions and sampling factors

As shown in Figure 2, a source image is defined to consist of **Nf** components. Each component, with unique identifier $\mathbf{C_i}$, is defined to consist of a rectangular array of samples of $\mathbf{x_i}$ columns by $\mathbf{y_i}$ lines. The component dimensions are derived from two parameters, **X** and **Y**, where **X** is the maximum of the $\mathbf{x_i}$ values and **Y** is the maximum of the $\mathbf{y_i}$ values for all components in the frame. For each component, sampling factors $\mathbf{H_i}$ and $\mathbf{V_i}$ are defined relating component dimensions $\mathbf{x_i}$ and $\mathbf{y_i}$ to maximum dimensions **X** and **Y**, according to the following expressions:

$$\mathbf{x_i} = \mathbf{X} \times \frac{\mathbf{H_i}}{\mathbf{H_{max}}} \quad \text{and} \quad \mathbf{y_i} = \left\lceil \mathbf{Y} \times \frac{\mathbf{V_i}}{\mathbf{V_{max}}} \right\rceil$$

where $\mathbf{H_{max}}$ and $\mathbf{V_{max}}$ are the maximum sampling factors for all components in the frame.

As an example, consider an image having 3 components with maximum dimensions of 512 lines and 512 columns, and with the following sampling factors:

- Component 0 $\mathbf{H_0} = 4,\ \mathbf{V_0} = 1$
- Component 1 $\mathbf{H_1} = 2,\ \mathbf{V_1} = 2$
- Component 2 $\mathbf{H_2} = 1,\ \mathbf{V_2} = 1$

Then $\mathbf{X} = 512$, $\mathbf{Y} = 512$, $\mathbf{H_{max}} = 4$, $\mathbf{V_{max}} = 2$, and $\mathbf{x_i}$ and $\mathbf{y_i}$ for each component are:

- Component 0 $\mathbf{x_0} = 512,\ \mathbf{y_0} = 256$
- Component 1 $\mathbf{x_1} = 256,\ \mathbf{y_1} = 512$
- Component 2 $\mathbf{x_2} = 128,\ \mathbf{y_2} = 256$

NOTE – The **X**, **Y**, $\mathbf{H_i}$, and $\mathbf{V_i}$ parameters are contained in the frame header of the compressed image data, whereas the individual component dimensions $\mathbf{x_i}$ and $\mathbf{y_i}$ are derived by the decoder. Source images with dimensions which do not satisfy the expressions above for $\mathbf{x_i}$ and $\mathbf{y_i}$ cannot be properly reconstructed.

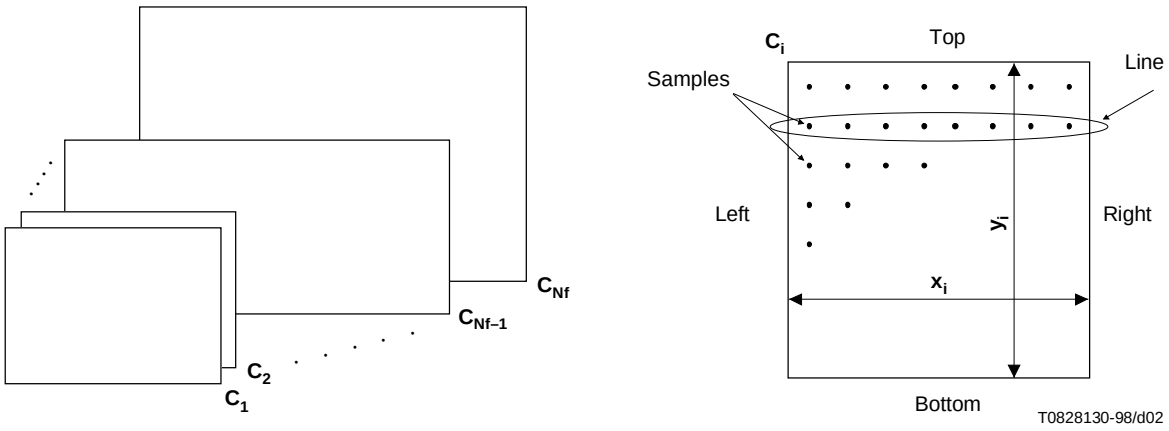
4.3.2 Sample precision and point transform

A sample is an integer with precision **P** bits, with any value in the range 0 through $2^{\mathbf{P}} - 1$. All samples of all components within an image shall have the same precision **P**. **P** is restricted to the range 2-16 bits.

Samples may optionally be divided by a power of 2 by a point transform applied prior to encoding. The point transform is an integer divide by $2^{\mathbf{Pt}}$, where **Pt** is the value of the point transform parameter. The output of the decoding process is re-scaled by multiplying by $2^{\mathbf{Pt}}$.

4.3.3 Orientation

Figure 2 indicates the orientation of an image component by the terms top, bottom, left, and right. The order by which the samples of an image component are input to the encoding procedures is defined to be left-to-right and top-to-bottom within the component. File formats (such as SPIFF, see 4.8.1) determine which edges of a source image are defined as top, bottom, left, and right.



a) Source image with multiple components

b) Characteristics of an image component

Figure 2 – Source image characteristics

4.4 Encoding process

4.4.1 Context modelling – Basics

The encoding process is described in outline below. The encoding process is specified in Annexes A, B, and D. An informative Annex, G, is included for additional explanation.

In this Recommendation | International Standard, the modelling approach used is based on the notion of "context". In context modelling, the encoding of each sample value is performed by conditioning on a small number of neighbouring samples. The context modelling procedure determines a probability distribution used to encode the current sample, whose position, *x*, is shown in Figure 3. The context is determined from four neighbourhood reconstructed samples at positions *a*, *b*, *c*, and *d* of the same component, as shown in Figure 3. From the values of the reconstructed samples at *a*, *b*, *c*, and *d*, the context first determines if the information in the sample *x* should be encoded in the regular or run mode:

- the run mode is selected when it is estimated from the context that successive samples are very likely to be nearly identical within the tolerances required for near-lossless coding (identical, for lossless coding);
- the regular mode is selected when it is estimated from the context that samples are not very likely to be nearly identical within the tolerances required for near-lossless coding (identical, for lossless coding).

4.4.2 Regular mode: Prediction and error encoding

In the regular mode, the context determination procedure is followed by a prediction procedure. The predictor combines the reconstructed values of the three neighbourhood samples at positions *a*, *b*, and *c* to form a predicted sample value at position *x* as shown in Figure 3. The prediction error is computed as the difference between the actual sample value at position *x* and its predicted value. This prediction error is then corrected by a context-dependent term to compensate for systematic biases in prediction. In the case of near-lossless coding, the prediction error is then quantized.

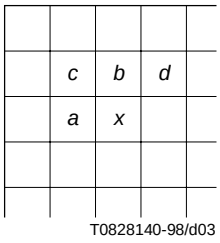


Figure 3 – Causal template used for context modelling and prediction

The corrected prediction error (further quantized for near-lossless coding) is then encoded using a procedure derived from Golomb coding (specified in Annex A and further described in Annex G).

NOTE – Golomb coding corresponds to Huffman coding for a geometric distribution.

The Golomb coding procedures specified in this Recommendation | International Standard depend on the context as well as prediction errors previously encoded for the same context.

4.4.3 Run mode

If the reconstructed values of the samples at *a*, *b*, *c*, and *d* are identical for lossless coding, or the differences between them (the local gradients, specified in Annex A) are within the bounds set for near-lossless coding, the context modelling procedure selects the run mode and the encoding process skips the prediction and error encoding procedures. In run mode, the encoder looks, starting at *x*, for a sequence of consecutive samples with values identical (or within the bound specified for near-lossless coding) to the reconstructed value of the sample at *a*. A run is ended by a sample of a different value (or one which exceeds the bound specified for near-lossless coding), or by the end of the current line, whichever comes first. The length information, which also specifies one of the above two run-ending alternatives, is encoded using a procedure specified in Annex A, which is extended from Golomb coding but has improved performance and adaptability.

4.5 Decoding process

The encoding and decoding processes are approximately symmetrical. Annex A specifies the encoding process, and Annex F describes the decoding process. The decoding process is followed by a sample mapping procedure which uses the value of each decoded sample as an index to a look-up table, provided in the compressed image data. The corresponding table entry might be of a different precision to that of the encoded sample.

NOTE – The sample mapping procedure is aimed at facilitating the use of this Recommendation | International Standard for the encoding of palletised and symbolic images. In the case of palletised images, the encoder would encode only one component, and the difference in precision would permit, for example, the display of single-component images as pseudo-colour ones. In the case of symbolic images, each decoded sample is a representation of the real image value, which is in the corresponding entry in the look-up table.

If no table is provided for a specific component, the output of the sample mapping procedure is identical to the input. The use of sample mapping is specified in Annex C.

4.6 Coding of multiple component images

The coding processes described in this Recommendation | International Standard can be applied to multiple components of an image, as well as to a single component image. Annex B describes how the coding processes shall be applied to images containing multiple components.

4.7 Compressed image data

The compressed image data output by the encoding process consists of marker segments and coded image data segments. The marker segments contain information required by the decoding process, including the image dimensions. The marker syntax is specified in Annex C, while the procedures for encoding the image data are specified in Annexes A, B, and D.

4.8 Interchange format

The interchange format is specified in Annex C based on Annex B of CCITT Rec. T.81 | ISO/IEC 10918-1. The interchange format allows a decoder to decode the compressed image data, regardless of specific application environments. Applications requiring further image information, for example for identifying an image to higher level applications, are recommended to use the SPIFF image file format, specified in Annex F of ITU-T Rec. T.84 | ISO/IEC 10918-3.

The following extension has been made to the SPIFF file format to allow its use with image data coded according to this Recommendation | International Standard:

4.8.1 Addition to SPIFF file header

In the file header described in Annex F.2.1 of ITU-T Rec. T.84 | ISO/IEC 10918-3, an additional value is allocated to parameter **C** describing the compression type. For data streams coded in accordance with this Recommendation | International Standard, **C** has the value X'06'. This has been standardised in ITU-T Rec. T.84 | ISO/IEC 10918-3 Amd.1, which also changes the version number **VERS** field in the SPIFF header to X'0200'.

5 Interchange format requirements

The interchange format is the coded representation of compressed image data for exchange between application environments.

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

6 Encoder requirements

An encoding process converts source image data (as defined in 4.2) to compressed image data (as defined in Annex C). Annexes A, B and D specify the encoding process.

An encoder is an embodiment of the encoding process. In order to conform with this Recommendation | International Standard, an encoder shall satisfy at least one of the following two requirements.

An encoder shall:

a) convert source image data to compressed image data which conform to the interchange format syntax specified in Annex C;

b) convert source image data to compressed image data which comply with the abbreviated format for compressed image data syntax specified in Annex C.

Conformance tests for the above requirements are specified in clause 8 of this Recommendation | International Standard.

NOTE – There is no requirement in this Recommendation | International Standard that any encoder which embodies the encoding process specified in Annexes A, B and D shall be able to operate for all ranges of the parameters which are allowed. An encoder is only required to meet the applicable conformance tests specified in clause 8, and to generate the compressed image data format according to Annex C for those parameter values which it does use.

7 Decoder requirements

A decoding process converts compressed image data to reconstructed image data. Since the decoding process is uniquely defined by the encoding process, there is no separate normative definition of the decoding process.

NOTE – The decoding process is not specified for non-compliant compressed image data.

A subsequent sample mapping procedure uses the value of each sample output by the decoding process as an index to map each sample value to an output sample value using the mapping tables specified for that sample component in Annex C. If no table is specified for that sample component, then the output sample value is identical to the sample value output by the decoding process. In this case, an inverse point transform may also be applied (see 4.3.2), thus completing a sample transformation process.

A decoder is an embodiment of the decoding process implicitly specified by the encoding process as specified in Annexes A, B and D, followed by the embodiment of the sample transformation process defined above. In order to conform to this Recommendation | International Standard, a decoder shall satisfy all three of the following requirements.

A decoder shall:

a) convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which comply with the interchange format syntax specified in Annex C. In the reconstructed image data output by the embodiment of the decoding process (before sample transformation), the value of each sample shall be identical to the reconstructed value defined in the encoding process specified in Annex A;

b) accept and properly store any table-specification data which conform to the abbreviated format for table-specification data syntax specified in Annex C;

c) convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which conforms to the abbreviated format for compressed image data syntax specified in Annex C, provided that the table specification data required for sample mapping has previously been installed in the decoder.

For the decoding process implicitly specified by the encoding process as specified in Annexes A, B and D, the conformance tests for the above requirements are specified in clause 8 of this Recommendation | International Standard.

NOTE – There is no requirement in this Recommendation | International Standard that any decoder which embodies the decoding process implicitly specified in Annexes A, B and D and the sample transformation process shall be able to operate for all ranges of the parameters which are allowed. A decoder is only required to meet the applicable conformance tests specified in clause 8, and to decode the compressed image data format specified in Annex C for those parameter values which it does use.

8 Conformance testing

8.1 Purpose

The conformance tests specified in this Recommendation | International Standard are intended to increase the likelihood of compressed image data interchange by specifying a range of tests for both encoders and decoders. The tests are not exhaustive tests of the respective functionality, and hence do not guarantee complete interoperability between independently implemented encoders and decoders. The main purpose of these conformance tests is to verify the validity of encoding and decoding process implementations, and the corresponding compressed image data. It is not an objective of these tests to carry out extensive verification of the interchange format or marker segment syntax. The marker segment syntax follows closely the interchange formats specified in Annex B of CCITT Rec. T.81 | ISO/IEC 10918-1, and testing procedures similar to those specified in ITU-T Rec. T.83 | ISO/IEC 10918-2 can be used for the purpose of verifying interchange format and marker segment syntax.

The tests are based on a set of test images which are incorporated into this specification in digital form. In addition, a reference encoder and decoder supporting the conformance tests specified in this Recommendation | International Standard (for the IBM PC) are incorporated into this specification in digital form.

8.2 Encoder conformance tests

Encoders are tested by encoding a source test image (see Annex E) using the encoder under test, and then decoding the compressed image data thus produced using a reference decoder. The image reconstructed by the reference decoder shall match the source image exactly in the case of lossless coding (parameter **NEAR** = 0, see Annex A). In the case of near-lossless coding (parameter **NEAR** > 0), the image reconstructed by the reference decoder shall match the image reconstructed by the reference decoder when fed with the compressed test image data.

The encode/decode cycle shall be carried out for each of the tests listed in Table E.2 using the test images listed in the "Source Image" column, and using the parameters specified in the "**NEAR**", "**ILV**", "Sub-sampling", and "Other JPEG-LS parameters" columns of the table. Restart markers shall not be inserted. The encoder testing procedure is illustrated in Figure 4.

NOTE 1 – An encoder can additionally be tested without a reference decoder by comparing the compressed image data produced by the encoder, for each of the tests in Table E.2, to the compressed test image data listed in Table E.2. This comparison shall be restricted to the coded data segments only, excluding marker segments (as different marker segments may represent the same coding parameters). The coded data segments produced by the encoder shall match those contained in the compressed test image data exactly. This procedure is illustrated in Figure 5.

NOTE 2 – The tests described in this clause represent minimal encoder conformance verification. More extensive encoder testing can be achieved by encoding arbitrary source images with the encoder under test, decoding the compressed image thus produced using a reference decoder, and comparing the image data reconstructed by the reference decoder with the original source image data. The image reconstructed by the reference decoder should match the source test image exactly in the case of lossless coding (parameter **NEAR** = 0, see Annex A). In the case of near-lossless coding (parameter **NEAR** > 0), the image reconstructed by the reference decoder should match the output image data generated by the application of an encoding / decoding cycle to the same source image.

The above conformance tests shall be performed without sample mapping and with **Pt** = 0.

8.3 Decoder conformance tests

Decoders are tested by decoding compressed test image data (see Annex E) using the decoder under test and comparing the reconstructed image to the corresponding source test image. The image reconstructed by the decoder under test shall exactly match the source test image in the case of lossless coding (**NEAR** = 0). In the case of near-lossless coding (**NEAR** > 0), the image reconstructed by the decoder under test shall match the image reconstructed by the reference decoder when fed with the compressed test image data.

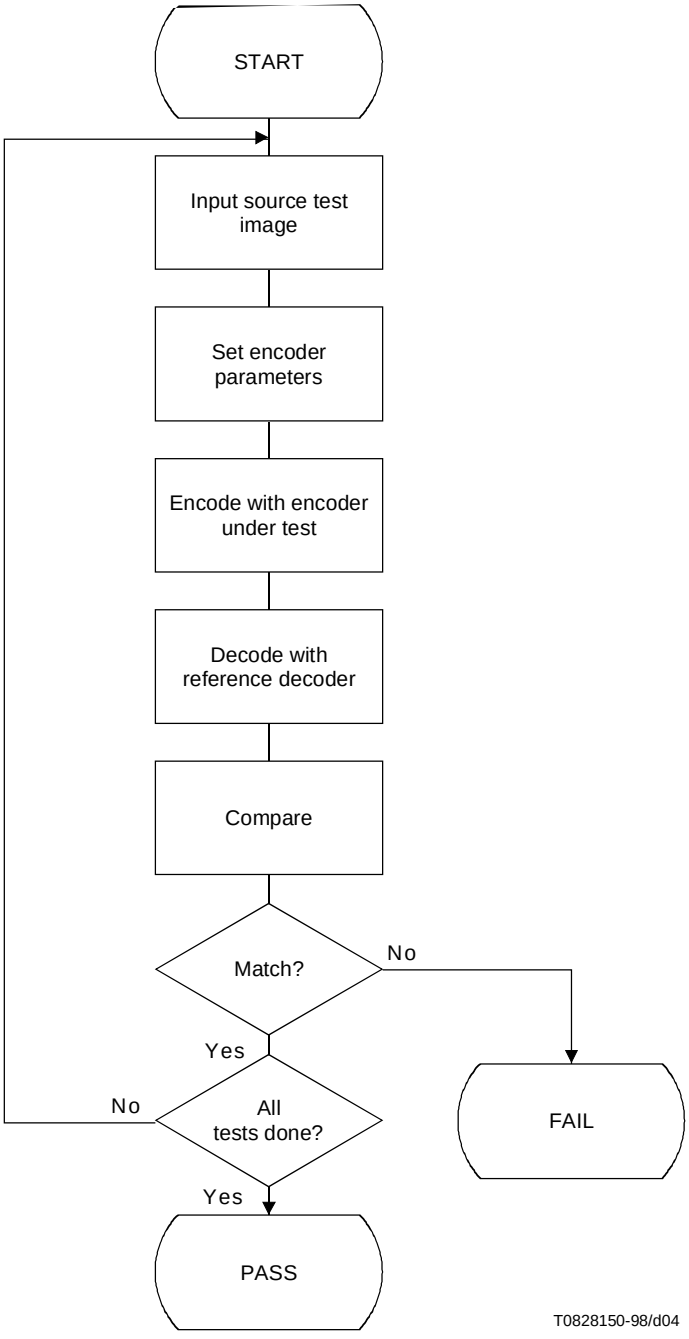


Figure 4 – Encoder testing with reference decoder

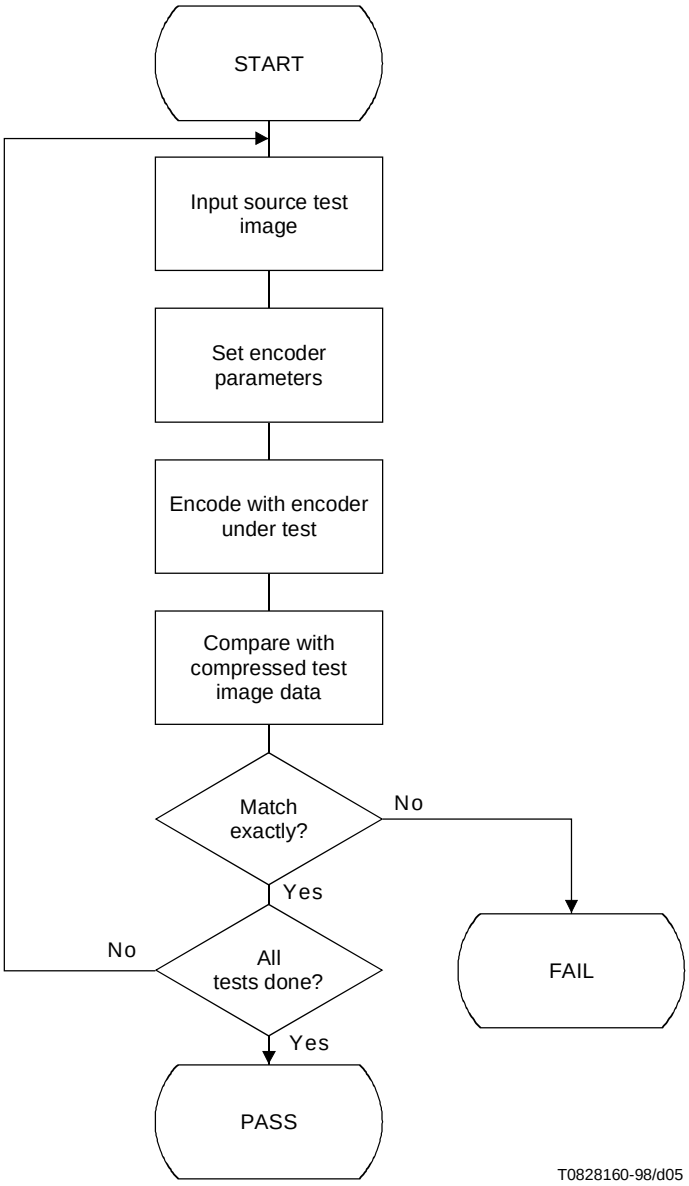


Figure 5 – Encoder testing without reference decoder

The decoding conformance tests shall be carried out for each of the tests listed in Table E.2, using as an input the compressed test image data listed in the "Compressed file name" column, with the parameters specified in the "**NEAR**", "**ILV**", "Sub-sampling", and "Other JPEG-LS parameters" columns of the table. The source test images used for the comparison are listed in the "Source image" column of Table E.2. In the case of lossless coding, no reference encoder or decoder is necessary for this decoder conformance test. The decoder testing procedure is illustrated in Figure 6.

NOTE – The tests specified represent minimal decoder conformance verification. More extensive decoder testing may be achieved by encoding arbitrary source image data with a reference encoder, decoding the compressed image data thus produced using the decoder under test and a reference decoder, and comparing the image data output by both decoders. The image data reconstructed by the decoder under test should match that reconstructed by the reference decoder for all samples.

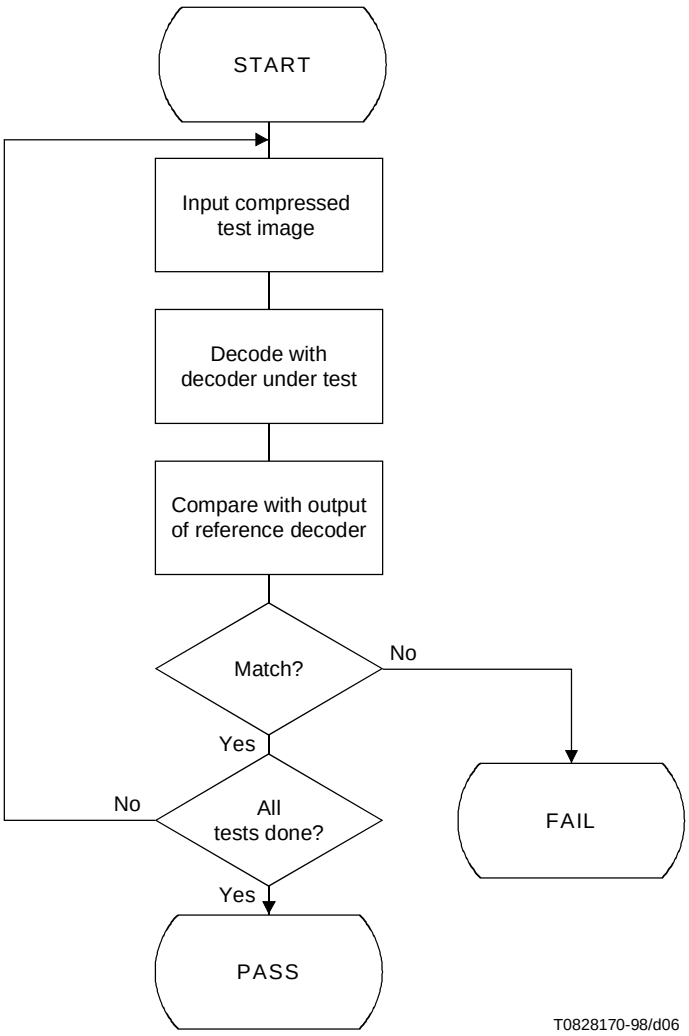


Figure 6 – Decoder testing procedure

Annex A

Encoding procedures for a single component

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

This annex specifies the encoding procedures defined by this Recommendation | International Standard. Clauses A.1 to A.7 define the encoding process. Clause A.8 summarises the provisions of this annex. The encoding procedures in this annex correspond to scans of a single component. The necessary modifications for dealing with multiple-component scans are specified in Annex B. Annex G (informative) includes a general description of the encoding process.

NOTE – There is **no requirement** in this Recommendation | International Standard that any encoder or decoder shall implement the procedures in precisely the manner specified in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or a decoder to be considered in conformance with this Recommendation | International Standard is that it satisfy the requirements determined by the conformance tests given in clause 8.

A.1 Coding parameters and compressed image data

A number of parameters are necessary to specify the coding process in this Recommendation | International Standard. The coding of these parameters in the compressed image data, and a normative set of default values for some of these parameters are specified in Annex C. This Recommendation | International Standard does not specify how these parameters are set in the encoding process by any application using it, if non-default values are used.

The bits generated by the encoding process forming the compressed image data shall be packed into 8-bit bytes. These bits shall fill bytes in decreasing order of significance. As an example, when outputting a binary code a_n, a_{n-1}, a_{n-2}, …. a_0, where a_n is the first output bit, and a_0 is the last output bit, a_n will fill the most significant available bit position in the currently incomplete output byte, followed by a_{n-1}, a_{n-2}, and so on. When an output byte is completed, it is placed as the next byte of the encoded bit stream, and a new byte is started. An incomplete byte, just before a marker, is padded with zero-valued bits before the insertion of any marker.

NOTE 1 – This padding differs from the padding method specified in CCITT Rec. T.81 | ISO/IEC 10918-1.

Marker segments are inserted in the data stream as specified in Annex D. In order to provide for easy detection of marker segments, a single byte with the value X'FF' in a coded image data segment shall be followed with the insertion of a single bit '0'. This inserted bit shall occupy the most significant bit of the next byte. If the X'FF' byte is followed by a single bit '1', then the decoder shall treat the byte which follows as the second byte of a marker, and process it in accordance with Annex C. If a '0' bit was inserted by the encoder, the decoder shall discard the inserted bit, which does not form part of the data stream to be decoded.

NOTE 2 – This marker segment detection procedure differs from the one specified in CCITT Rec. T.81 | ISO/IEC 10918-1.

A.2 Initialisations and conventions

A.2.1 Initialisations

The context modelling procedure specified in this annex uses the causal template *a, b, c* and *d* depicted in Figure 3. When encoding the first line of a source image component, the samples at positions *b, c*, and *d* are not present, and their reconstructed values are defined to be zero. If the sample at position *x* is at the start or end of a line so that either *a* and *c*, or *d* is not present, the reconstructed value for a sample in position *a* or *d* is defined to be equal to *Rb*, the reconstructed value of the sample at position *b*, or zero for the first line in the component. The reconstructed value at a sample in position *c*, in turn, is copied (for lines other than the first line) from the value that was assigned to *Ra* when encoding the first sample in the previous line.

The following initialisations shall be performed at the start of the encoding process of a scan, as well as in other situations specified in Annex D. All variables are defined to be integers with sufficient precision to allow the execution of the required arithmetic operations without overflow or underflow, given the bounds on the parameters indicated in Annex C:

1) Compute the parameter **RANGE**: For lossless coding (**NEAR** = 0), **RANGE = MAXVAL + 1**. For near-lossless coding (**NEAR** > 0):

$$\mathbf{RANGE} = \left\lfloor \frac{\mathbf{MAXVAL} + 2*\mathbf{NEAR}}{2*\mathbf{NEAR} + 1} \right\rfloor + 1$$

NOTE – **MAXVAL** and **NEAR** are coding parameters whose values are either default or set by the application (see Annex C).

Compute the parameters **qbpp** = log **RANGE** , **bpp** = max(2, ⌈log(**MAXVAL**+1)), and **LIMIT** = 2 * (**bpp** + max(8,**bpp**)).

2) Initialise the variables *N*[0..366], *A*[0..366], *B*[0..364] and *C*[0..364], where the nomenclature [0..*i*] indicates that there are *i*+1 instances of the variable. The instances are indexed by [*Q*], where *Q* is an integer between 0 and *i*. The indexes [0..364] correspond to the regular mode contexts (total of 365, see A.3), whilst the indexes [365] and [366] correspond to run mode interruption contexts. For example, *C*[5] corresponds to the variable *C* in the regular mode context indexed by 5. Each one of the entries of *A* is initialised with the value

$$\max\left(2, \left\lfloor \frac{\mathbf{RANGE} + 2^5}{2^6} \right\rfloor\right)$$

those of *N* are initialised with the value 1, and those of *B* and *C* with the value 0.

3) Initialise the variables for the run mode: *RUNindex*=0 and **J**[0..31] = {0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9,10,11,12,13,14,15}.

4) Initialise the two run interruption variables *Nn*[365] and *Nn*[366] to 0.

A.2.2 Conventions for code segments

In the remaining clauses of this annex, various procedures of the encoding process are specified in software code segments, written in the C programming language, as specified in ISO/IEC 9899. The syntax and semantics of C shall be assumed in all code segments contained in this annex.

All variables used in the code segments are assumed to be integer, and to have sufficient precision to allow the execution of the required arithmetic operations without overflow or underflow, given the bounds on the parameters indicated in Annex C. When division and right shift operations are indicated, all variables used are non-negative integers so that the exact computation of quotients, remainders and shifted quantities is unambiguously specified. The code segments are used to specify parts of the encoding process, and do not constitute, by themselves or in any aggregation, a full implementation of the process.

In addition to the variables and parameters specified in 3.1 for the encoding and decoding processes, the following auxiliary labels, global variables, and functions are used in the software code segments:

abs(*i*)	Function: returns the absolute value of *i* in accordance with the definition in 3.1.3
RunModeProcessing	Label: indicates the beginning of the run mode process as defined in A.7
RegularModeProcessing	Label: indicates the gradient quantization step as defined in A.3.3
max(*i*, *j*)	Function: returns the maximum of *i* and *j* in accordance with the definition in 3.1.47
min(*i*, *j*)	Function: returns the minimum of *i* and *j* in accordance with the definition in 3.1.48

GetNextSample()	Function: reads the next sample in the source image and sets the corresponding values of *x, a, b, c, d, Ix, Ra, Rb, Rc, Rd*. In multi-component images, the concept of *next sample* depends on the interleaving mode used, as described in Annex B. If the sample read is at the end of the current image line, **GetNextSample()** sets the "global" variable *EOLine* to 1. In all other cases, *EOLine* is reset to 0. When processing samples in regular mode, it is implicitly assumed that **GetNextSample()** is invoked before the processing of each sample. Invocation of **GetNextSample()** is shown explicitly in the description of the run mode in A.7, due to the special treatment required at ends of lines in that mode. The reconstructed values *Ra, Rb, Rc,* and *Rd* inherit their value from a previous computation of a reconstructed value *Rx* as described in A.4.4, A.7.1, and A.7.2.
EOLine	Global variable: set by **GetNextSample()**: equal to 1 if the current sample is the last in the line, 0 otherwise.
AppendToBitStream(*i*, *j***)**	Function: appends the non-negative number *i* in binary form to the encoded bit stream, using *j* bits. Most significant bits are appended first. The process guarantees that *j* bits are sufficient to represent *i* exactly.
Quantize(*i***)**	Function: returns the quantized value of *i* following the procedure applied to *Errval* in Code segment A.8 ("if" statement). This function is used to quantize the prediction error in near-lossless coding.
ModRange(*i*, **RANGE)**	Function: returns the value of *i* modulo **RANGE** as described in A.4.5.
ComputeRx()	Function: returns the reconstructed value *Rx* of the current sample as described in Code segment A.8 (after the "if" statement). This function reconstructs the value of *Rx* from the quantized prediction error.

A.3 Context determination

After a number of samples have been coded scanning from left to right and from top to bottom, the sample *x* positioned as in Figure 3 shall be encoded. The context at this sample shall be determined by the previously reconstructed values *Ra*, *Rb*, *Rc*, and *Rd* corresponding to the samples *a*, *b*, *c*, and *d* as shown in Figure 3**,** respectively. In lossless coding, the reconstructed values are identical to those of the source image data. The steps in context determination, to be performed in the presented order, are the following:

A.3.1 Local gradient computation

The first step in the context determination procedure shall be to compute the local gradient values, *D1*, *D2*, *D3* of the neighbourhood samples, as indicated in Code segment A.1.

Code segment A.1 – Local gradient computation for context determination

```
D1 = Rd – Rb;
D2 = Rb – Rc;
D3 = Rc – Ra;
```

A.3.2 Mode selection

If the local gradients are all zero (for lossless coding), or their absolute values are less than or equal to **NEAR**, the allowed error for near-lossless coding, the encoder shall enter the run mode, otherwise the encoder shall enter the regular mode. The mode selection procedure is specified in Code segment A.2. In the case of lossless coding, this mode selection procedure is equivalent to the procedure shown in Code segment A.3, where the encoder is checking if *Ra=Rb=Rc=Rd*.

Code segment A.2 – Mode selection procedure

```
if ((abs(D1) <= NEAR) && (abs(D2) <= NEAR) && (abs(D3) <= NEAR))
        goto RunModeProcessing
else
        goto RegularModeProcessing;
```

Code segment A.3 – Mode selection procedure for lossless coding

```
if (D1 == 0 && D2 == 0 && D3 == 0)
        goto RunModeProcessing;
else
        goto RegularModeProcessing;
```

If run mode is selected, the encoding process shall proceed as specified in A.7. Clauses A.3.3 to A.6.2 apply only to regular mode.

A.3.3 Local gradient quantization

The context determination procedure shall continue by quantizing *D1*, *D2*, and *D3* according to the procedure specified in Code segment A.4. For this purpose, non-negative thresholds, **T1**, **T2**, and **T3,** are used. The default values of these thresholds – and ways to explicitly override these defaults – are specified in C.2.4.1. In Code segment A.4, the entry *Di* to the procedure is one of the values *D1*, *D2*, or *D3* from the local gradient computation step. According to their relation with the thresholds, a region number *Qi* is obtained (*Q1*, *Q2*, and *Q3* respectively). This forms a vector (*Q1*, *Q2*, *Q3*) representing the context for the sample *x*. Since there are 9 quantization regions for each of the gradients, *Q1*, *Q2*, and *Q3*, each is allocated one of nine possible numbers between –4 and 4.

Code segment A.4 – Quantization of the gradients

```
if (Di <= –T3) Qi = –4;
else if (Di <= –T2) Qi = –3;
else if (Di <= –T1) Qi = –2;
else if (Di < – NEAR) Qi = –1;
else if (Di <=  NEAR) Qi = 0;
else if (Di <  T1) Qi = 1;
else if (Di <  T2) Qi = 2;
else if (Di <  T3) Qi = 3;
else Qi = 4;
```

A.3.4 Quantized gradient merging

If the first non-zero element of the vector (*Q1*, *Q2*, *Q3*) is negative, then all the signs of the vector (*Q1*, *Q2*, *Q3*) shall be reversed to obtain (*–Q1*, *–Q2*, *–Q3*).

In this case, the variable *SIGN* shall be set to –1, otherwise it shall be set to +1. After this possible "merging", the vector (*Q1*, *Q2*, *Q3*) is mapped, on a one-to-one basis, into an integer *Q* representing the context for the sample *x*. The function mapping the vector (*Q1*, *Q2*, *Q3*) to the integer *Q* is not specified in this Recommendation | International Standard. This Recommendation | International Standard only requires that the mapping shall be one-to-one, that it shall produce an integer in the range [0..364], and that it be defined for all possible values of the vector (*Q1*, *Q2*, *Q3*), including the vector (0, 0, 0).

NOTE – A total of $9 \times 9 \times 9 = 729$ possible vectors are defined by the procedure in Code segment A.4. The vector (0, 0, 0) and its corresponding mapped value can only occur in regular mode for sample interleaved multi-component scans, as detailed in Annex B.

A.4 Prediction

This prediction procedure is performed only in the regular (non-run) mode. The following steps shall be performed in the order specified.

A.4.1 Edge-detecting predictor

An estimate *Px* of the value at the sample at *x* to be encoded shall be determined from the values *Ra*, *Rb*, and *Rc* at the positions *a*, *b*, and *c* specified in Figure 3, as indicated in Code segment A.5.

Code segment A.5 – Edge-detecting predictor

```
if (Rc >= max(Ra, Rb))
    Px = min(Ra, Rb);
else {
    if (Rc <= min(Ra, Rb))
        Px = max(Ra, Rb);
    else
        Px = Ra + Rb – Rc;
}
```

A.4.2 Prediction correction

After *Px* is computed, the prediction shall be corrected according to the procedure depicted in Code segment A.6, which depends on *SIGN*, the sign detected in the context determination procedure. The new value of *Px* shall be clamped to the range [0..**MAXVAL**]. The prediction correction value *C*[*Q*] is derived from the bias as specified in A.6.2.

Code segment A.6 – Prediction correction from the bias

```
if (SIGN == +1)
    Px = Px + C[Q];
else
    Px = Px – C[Q];
if (Px > MAXVAL)
    Px = MAXVAL;
else if (Px < 0)
    Px = 0;
```

A.4.3 Computation of prediction error

Using the value of *Px*, corrected by the above procedure, the prediction error, *Errval*, shall be computed. If the sign of the context, given by *SIGN*, is negative, the sign of the error shall be reversed. This is shown in Code segment A.7 for the sample at position *x*, with value *Ix*.

Code segment A.7 – Computation of prediction error

```
Errval = Ix – Px;
if (SIGN == –1)
    Errval = – Errval;
```

A.4.4 Error quantization for near-lossless coding, and reconstructed value computation

In lossless coding (**NEAR** = 0), the reconstructed value *Rx* shall be set to *Ix*. In near-lossless coding (**NEAR** > 0), the error shall be quantized. After quantization, the reconstructed value *Rx* of the sample *x*, which is used to encode further samples, shall be computed in the same manner as the decoder computes it. These operations are shown in Code segment A.8.

Code segment A.8 – Error quantization and computation of the reconstructed value in near-lossless coding

```
if (Errval > 0)
    Errval = (Errval + NEAR) / (2 * NEAR + 1);
else
    Errval = – (NEAR – Errval) / (2 * NEAR + 1);
Rx = Px + SIGN * Errval * (2 * NEAR + 1);
if (Rx < 0)
    Rx = 0;
else if (Rx > MAXVAL)
    Rx = MAXVAL;
```

A.4.5 Modulo reduction of the prediction error

The error shall be reduced to the range relevant for coding, $(-\mathbf{RANGE}/2 \;..\; \lceil \mathbf{RANGE}/2 \rceil - 1)$. This is achieved with the steps detailed in Code segment A.9 (function **ModRange**()).

Code segment A.9 – Modulo reduction of the error

```
if (Errval < 0)
        Errval = Errval + RANGE;
if (Errval >= ((RANGE + 1) / 2))
        Errval = Errval – RANGE;
```

A.5 Prediction error encoding

The next step of the regular mode shall be to encode the error. For this, the variables *A*[0..364] and *N*[0..364] are used to compute the Golomb coding variable *k.* The computation of the variable *k* is context-dependent and shall be performed as indicated in Code segment A.10. The variable *Errval* shall then be mapped to a non-negative integer, *MErrval*, and encoded using the code function *LG*(*k*, **LIMIT**) (Annex G contains an informative description of this procedure).

A.5.1 Golomb coding variable computation

The variable *k*, for the limited length Golomb code function *LG*(*k*, **LIMIT**), shall be computed by the procedure indicated in Code segment A.10. This variable is context-dependent.

Code segment A.10 – Computation of the Golomg coding variable *k*

```
for(k=0; (N[Q]<<k)<A[Q]; k++);
```

A.5.2 Error mapping

The prediction error, *Errval* shall be mapped to a non-negative value, *MErrval* as specified in Code segment A.11. For lossless coding, the mapping procedure checks the value of *k* (the Golomb coding variable) and according to its value performs a "regular mapping" ($k \neq 0$), or a "special mapping" ($k = 0$ and $B[Q]$ less or equal than $-N[Q]/2$), which is equivalent to encoding $-(Errval+1)$ with the regular mapping. For near-lossless coding, the mapping is independent of the value of *k.*

Code segment A.11 – Error mapping to non-negative values

```
if ((NEAR == 0) && ( k == 0) && ( 2 * B[Q] <= – N[Q])) {
        if (Errval >= 0)
                MErrval = 2 * Errval + 1
        else
                MErrval = –2 * (Errval + 1);
}
else {
        if (Errval >= 0
                MErrval = 2 * Errval;
        else
                MErrval = –2 * Errval – 1;
}
```

A.5.3 Mapped-error encoding

The mapped error value, *MErrval*, shall be encoded with the limited length Golomb code function *LG*(*k*, **LIMIT**) defined by the following procedure:

1. If the number formed by the high order bits of *MErrval* (all but the *k* least significant bits) is less than **LIMIT** – **qbpp** – 1, this number shall be appended to the encoded bit stream in unary representation, that is, by as many zeros as the value of this number, followed by a binary one. The *k* least significant bits of *MErrval* shall then be appended to the encoded bit stream without change, with the most significant bit first, followed by the remaining bits in decreasing order of significance.

2. Otherwise, **LIMIT** – **qbpp** – 1 zeros shall be appended to the encoded bit stream, followed by a binary one. The binary representation of *MErrval* – 1 shall then be appended to the encoded bit stream using **qbpp** bits, with the most significant bit first, followed by the remaining bits in decreasing order of significance.

A.6 Update variables

The last step of the encoding of the sample *x* in the regular mode is the update of the variables *A*, *B*, *C*, and *N*. It is important to note that this update shall be performed at the end of the coding procedure, after *k* and *MErrval* are computed.

A.6.1 Update

The variables *A*[*Q*], *B*[*Q*], and *N*[*Q*] are updated according to the current prediction error, as in Code segment A.12. The variables *A*[*Q*] and *B*[*Q*] accumulate prediction error magnitudes and values for context *Q*, respectively. The variable *N*[*Q*] accounts for the number of occurrences of the context *Q* since initialisation.

Code segment A.12 – Variables update

```
B[Q] = B[Q] + Errval *(2 *NEAR + 1);
A[Q] = A[Q] + abs(Errval);
if (N[Q] == RESET) {
        A[Q] == A[Q] >> 1;
        if (B[Q] >= 0)
                B[Q] = B[Q] >> 1;
        else
                B[Q] = –((1-B[Q]) >> 1);
        N[Q] = N[Q] >> 1;
}
  N[Q] = N[Q] + 1;
```

NOTE – In lossless coding, the value added to *B*[*Q*] is the signed error, after modulo reduction.

RESET is a JPEG-LS coding parameter whose value is either default or set by the application (see Annex C).

A.6.2 Bias computation

The bias variable *B*[*Q*] allows an update of the prediction correction value *C*[*Q*] by at most one unit every iteration. The variables are clamped to limit their range of possible values. The prediction correction value *C*[*Q*] shall be computed according to the procedure in Code segment A.13**,** which also yields an update of *B*[*Q*].

Code segment A.13 – Update of bias-related variables *B*[*Q*] **and** C[*Q*]

```
if (B[Q] <= –N[Q]) {
                B[Q] = B[Q] + N[Q];
                if (C[Q] > MIN_C)
                   C[Q] = C[Q – 1;
                if (B[Q] <= –N[Q])
                   B[Q] = –N[Q] + 1;
}
else if (B[Q] > 0) {
                B[Q] = B[Q] – N[Q];
                if C[Q] < MAX_C)
                        C[Q] = C[Q] + 1;
                if (B[Q] > 0)
                        B[Q] = 0
}
```

The constants **MIN_C** and **MAX_C** are defined in 3.3.

A.7 Run mode

If the local gradients are all equal to zero (for lossless coding), or their absolute values are less than or equal to **NEAR** (for near-lossless coding), then the process shall enter run mode (see A.3.2). In lossless coding, the encoder shall read subsequent samples into *Ix* while *Ix* equals the reconstructed value *Ra*, which refers to the corresponding sample *a* at the beginning of the run, or the end of the current image line is encountered. In near-lossless coding, if the absolute value of the difference between *Ix* and *Ra* is less than or equal to the allowed error (**NEAR**), the run continues. The encoding of the run length shall be followed by the encoding of the last scanned sample (i.e. run interruption sample) in case the run is interrupted other than by the end of the current image line being encountered.

The run mode procedure is composed of two main steps: run scanning and run-length coding; and run interruption coding.

A.7.1 Run scanning and run-length coding

Given a "code-order" *rg*, where *rg* is restricted to a *rk*-th power of 2, a one bit code word '1', is used to encode run segments of length *rg* as well as shorter segments that were interrupted by the end of a line. A *rk*+1 bit code word shall be used to encode any remaining run segment. The first case represents a "hit" situation, where a run of length *rg* is achieved except at the end of a line, while the second case is a "miss" situation, where the run is interrupted before achieving the "maximal" segment length *rg*. In this miss situation, a prefix bit, '0', shall be sent, followed by the actual length of the remaining run segment, which shall be encoded with *rk* bits. The value of *rg* shall be adapted, according to a pre-defined table **J** of 32 entries for values of *rk* (see A.2.1, Step 3) , each time a run segment of length *rg* is scanned (the index to the table **J** increases) or a miss has occurred (the index to the table **J** decreases). The applicable procedures are given below.

NOTE – The following description of the run scanning and coding procedures suggests an implementation in which the run length is encoded only after detecting the termination of the run. However, it is possible to start encoding the run length as soon as a run of length *rg* is detected.

A.7.1.1 Run scanning

The first step in the run mode is to read the source image data into *Ix* and determine a run length, *RUNcnt*. This is specified as indicated in Code segment A.14.

Code segment A.14 – Run-length determination for run mode

```
RUNval = Ra;
RUNcnt = 0;
while (abs(Ix – RUNval) <= NEAR) {
        RUNcnt = RUNcnt + 1;
        Rx = RUNval;
        if (EOLine == 1)
           break;
else
              GetNextSample();
}
```

NOTE – The test abs(Ix – RUNval) <= **NEAR** reduces, in the lossless case, to *Ix* == *RUNval*.

A.7.1.2 Run-length coding

The variable *RUNcnt* computed following the procedure in Code segment A.14 represents the run length. The next step is to encode this number. A '1' shall be appended to the bit stream for each run of length *rg*, where *rg* shall be obtained from the 32-entries table **J**. The index, *RUNindex*, to the table **J** shall be increased by 1, up to a maximum value of 31, each time a run of length *rg* is reached. The table **J** contains values for *rk*, not *rg*. The complete procedure for this part is specified in Code segment A.15. If the run is interrupted by an end of line (setting *EOLine* = 1), and the remaining length after successive subtractions of *rg* is greater than zero, an extra '1' shall be appended to the bit stream. Else, if the run was interrupted by a sample of a different value, the remaining length is coded by a code word of length *rk*+1 (a prefix bit, '0', followed by *rk* bits to encode the remaining run length), and the index *RUNindex* is decreased by 1 (not to less than 0). This is detailed in Code segment A.16.

Code segment A.15 – Encoding of run segments of length *rg*

```
while (RUNcnt >= (1 << J[RUNindex]) ) {
        AppendToBitStream(1,1);
        RUNcnt = RUNcnt – (1 << J[RUNindex]);
        if (RUNindex < 31)
                RUNindex = RUNindex +1;
}
```

Code segment A.16 – Encoding of run segments of length less than *rg*

```
if (abs(Ix – RUNval) > NEAR) {
        AppendToBitStream(0,1);
        AppendToBitStream(RUNcnt, J[RUNindex]);
        if (RUNindex > 0)
                RUNindex = RUNindex –1;
}
else if (RUNcnt > 0)
        AppendToBitStream(1,1);
```

A.7.2 Run interruption sample encoding

If the run is interrupted other than by the end of the image line, the new sample that caused the run interruption shall be encoded. This shall be done by encoding the difference between the value *Ix* at the current position *x*, and the reconstructed value at *a* or *b* (both positions relative to *x*). In this mode of operation, two different contexts are used: The first is when the absolute value of the difference between *Ra* and *Rb* is not larger than **NEAR**, and the second when this absolute value is larger than **NEAR**.

NOTE – The basic concepts in the run interruption encoding are the same as those used to encode a new sample in the regular encoding mode, with the additional requirement that *Ix* must differ from *Ra* by more than **NEAR**, otherwise the run would have continued.

The following actions shall be carried out:

1. Compute the index *RItype* as indicated in Code segment A.17. This index defines a context in this mode, similar to the variable *Q* in regular mode.

Code segment A.17 – Index computation

```
if (abs(Ra – Rb) <= NEAR
        RItype = 1;
else
        RItype = 0;
```

2. Compute the prediction error, as indicated in Code segment A.18

Code segment A.18 – Prediction error for a run interruption sample

```
if (RItype ==1)
        Px = Ra;
else
        Px = Rb
Errval = Ix – Px;
```

3. Correct, if necessary, the sign of *Errval* (see Code segment A.19). This step is analogous to the context-merging procedure in the regular coding mode. For near-lossless coding, *Errval* shall be quantized and *Rx* computed, as shown in Figure A.8. The error shall then be reduced using the variable **RANGE**, following the same steps as in A.4.5 (this reduction is performed by the function **ModRange** below).

Code segment A.19 – Error computation for a run interruption sample

```
if ((RItype == 0) && (Ra > Rb)) {
        Errval = –Errval;
        SIGN = –1;
}
else
        SIGN = 1;
if (NEAR > 0) {
        Errval = Quantize(Errval);
        Rx = ComputeRx ();
}
else
        Rx = Ix;
Errval = ModRange (Errval,RANGE);
```

4. Compute the auxiliary variable *TEMP*. This variable is used for the computation of the Golomb variable *k*.

Code segment A.20 – Computation of the auxiliary variable TEMP

```
if (RItype == 0)
        TEMP = A[365];
else
        TEMP = A[366] + (N[366] >> 1);
```

5. Set $Q = RItype + 365$. The Golomb variable *k* shall be computed, following the same procedure as in the regular mode, Code segment A.10, but using *TEMP* instead of *A*[*Q*].

6. Compute the flag *map*, as indicated in Code segment A.21. This variable influences the mapping of *Errval* to non-negative values, as indicated in Code segment A.22.

Code segment A.21 – Computation of *map* for *Errval* mapping

```
if ((k == 0) && (Errval > 0) && (2 * Nn[Q] < N[Q]))
        map = 1;
else if ((Errval < 0) && (2 * Nn[Q] >= N[Q]))
        map = 1;
else if ((Errval < 0) && (k ! = 0))
        map = 1;
else
        map = 0;
```

7. *Errval* is now mapped:

Code segment A.22 – *Errval* mapping for run interruption sample

```
EMErrval = 2 * abs(Errval) – RItype – map;
```

8. Encode *EMErrval* following the same procedures as in the regular mode (see A.5.3), but using the limited length Golomb code function *LG*(*k, glimit*), where *glimit* = **LIMIT** – **J**[*RUNindex*] – 1 and *RUNindex* corresponds to the value of the variable before the decrement specified in Code segment A.16.

9. Update the variables for run interruption sample encoding, according to Code segment A.23.

Code segment A.23 – Update of variables for run interruption sample

```
if Errval < 0)
        Nn[Q] = Nn[Q] + 1;
A[Q] = A[Q] + ((EMErrval + 1 RItype) >> 1);
if (N[Q] == RESET) {
        A[Q] = A[Q] >> 1;
        N[Q] = N[Q] >> 1;
        Nn[Q] = Nn[Q] >> 1;
}
N[Q] = N[Q] + 1;
```

A.8 Flow of encoding procedures

The order in which the encoding procedures shall be performed is summarised below.

1. Initialisation:
   a) Assign default parameter values to JPEG-LS preset coding parameters not specified by the application (see A.1).
   b) Initialise the non-defined samples of the causal template (see A.2.1).
   c) Compute the parameter **RANGE** (see A.2.1): For lossless coding, **RANGE** = **MAXVAL** + 1. For near-lossless coding

$$\mathbf{RANGE} = \left\lfloor \frac{\mathbf{MAXVAL} + 2*\mathbf{NEAR}}{2*\mathbf{NEAR} + 1} \right\rfloor + 1 .$$

   Compute the parameters **qbpp** = $\lceil \log \mathbf{RANGE} \rceil$, **bpp** = max(2,$\lceil \log(\mathbf{MAXVAL} + 1) \rceil$), and **LIMIT** = 2 * (**bpp** + max(8, **bpp**)).
   d) For each context Q, initialise four variables (see A.2.1): $A[Q] = \max\left(2, \left\lfloor \frac{\mathbf{RANGE} + 2^5}{2^6} \right\rfloor\right)$, $B[Q] = C[Q] = 0$, $N[Q] = 1$. For $A[Q]$ and $N[Q]$, Q is an integer between 0 and 366; for $B[Q]$ and $C[Q]$, Q is an integer between 0 and 364 (regular mode contexts only).
   e) Initialise the variables for the run mode procedure: *RUNindex*=0 and **J**[0..31] = {0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9,10,11,12,13,14,15}.
   f) Initialise the two run interruption variables *Nn*[365] and *Nn*[366] to 0 (see A.2.1).
   g) Set current sample to the first sample in the source image.
2. For the current sample, compute the local gradients according to Code segment A.1.
3. Select the coding mode following the procedure in Code segment A.2. If run mode is selected, go to Step 17, otherwise continue with the regular mode.
4. Quantize the local gradients according to the steps detailed in Code segment A.4.
5. Check and change if necessary the signs of the components of the vector representing the context, modifying accordingly the variable *SIGN* (see A.3.4).
6. Compute *Px* according to Code segment A.5.
7. Correct *Px* using *C*[*Q*] and the variable *SIGN*, and clamp the corrected value to the interval [0..**MAXVAL**] according to the procedure in Code segment A.6.
8. Compute the prediction error and, if necessary, invert its sign according to the procedure in Code segment A.7.
9. For near-lossless coding, quantize the error and compute the reconstructed value of the current sample according to Code segment A.8. For lossless coding, update the reconstructed value by setting *Rx* equal to *Ix*.

10. Reduce the error to the relevant range according to Code segment A.9.
11. Compute the context-dependent Golomb variable *k* according to the procedure in Code segment A.10.
12. Perform the error mapping according to the procedure in Code segment A.11.
13. Encode the mapped error value *MErrval* using the limited length Golomb code function *LG*(*k*, **LIMIT**), as specified in A.5.3.
14. Update the variables according to Code segment A.12.
15. Update the prediction correction value *C*[*Q*] according to the procedure in Code segment A.13.
16. Go to step 2 to process the next sample.
17. Run mode coding:
    a) Set *RUNval = Ra*. While (abs(*Ix – RUNval)* <= **NEAR**), increment *RUNcnt*, and if not at the end of a line, read a new sample. Set *Rx = RUNval* each time the sample *x* is added to the run (see Code segment A.14).
    b) While $RUNcnt \geq 2^{\mathbf{J}[RUNIndex]}$, do (see Code segment A.15):
        i) Append '1' to the bit stream.
        ii) $RUNcnt = RUNcnt - 2^{\mathbf{J}[RUNIndex]}$.
        iii) If *RUNindex* < 31, then increment *RUNindex* by one.
    c) If the run was interrupted by the end of a line (see Code segment A.16):
        i) If *RUNcnt* > 0, append '1' to the bit stream.
        ii) Go to Step 16.
    d) Append '0' to the bit stream (see Code segment A.16).
    e) Append *RUNcnt* in binary representation (using **J**[*RUNindex*] bits) to the bit stream (MSB first, see Code segment A.16).
    f) If *RUNindex* > 0, then decrement *RUNindex* by one (see Code segment A.16).
18. Run interruption sample encoding: perform the operations in A.7.2, and then go to Step 16.

Annex B

Multi-component images

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

B.1 Introduction

For encoding images with more than one component (e.g. colour images), this Recommendation | International Standard supports combinations of single-component scans and multi-component scans, as specified in Annex C. For multi-component scans, two modes (described below) are supported: *line interleaved* and *sample interleaved*. The specific components per scan are specified in the scan header (see Annex C), as well as the interleave mode (as specified by parameter **ILV**), which describes the structure within a single scan. The parameter **ILV** admits the values 0 (non-interleaved), 1 (line interleaved) and 2 (sample interleaved).

For multi-component scans, a single set of context counters (*A, B, C, N* and *Nn*) is used across all the components in the scan. The prediction and context modelling procedures shall be performed as in the single-component case, and are component independent, meaning that samples from one component are not used to predict or compute the context of samples from another component.

All the encoding and decoding variables (e.g. *A*[*0..366*]) shall be set to their initial values, as described in Annex A, when a new scan is to be encoded (starting from Step 1 in A.8). The dimensions of each component are given by the information in the frame header. The byte completion padding described in A.1 applies also to multi-component scans.

B.2 Line interleaved mode

B.2.1 Description

This mode is specified by setting the parameter **ILV** in the start of scan marker segment to a value of 1. In this mode, for each component $\mathbf{C_i}$ in a scan, a set of $\mathbf{V_i}$ consecutive lines is encoded before starting the encoding of $\mathbf{V_{i+1}}$ lines of the subsequent component $\mathbf{C_{i+1}}$. The values $\mathbf{V_i}$ are specified in the start of frame marker segment as vertical sampling factors, see Annex C. For a scan with **Ns** components, the number of lines interleaved and encoded follows the sequence

$\mathbf{V_1\ V_2 \ldots V_{Ns}}$, $\mathbf{V_1\ V_2 \ldots V_{Ns}}$, $\mathbf{V_1\ V_2 \ldots V_{Ns}}$, etc.

The value of the variable *RUNindex* for the run mode is component dependent, one value of the variable being used for each component. The prediction and context determination procedures shall be performed as in the single-component mode, and do not use information from the multiple components. Except for the first line of the first component, which is always coded at the beginning of a new byte in the encoded bit stream, there is no byte alignment between encoded lines, and encoded lines can start and end at any bit position in the byte.

B.2.2 Process flow

The process flow for the line interleaved encoding mode is specified below, in terms of the general process flow given in A.8. For convenience, the steps are numbered identically to those in A.8.

1. Initialise a single set of variables as in Step 1 in the procedure described in A.8, except for the run mode variable *RUNindex*, for which one copy per component is initialised.

2. Follow Steps 2-15 in A.8 for the current sample in the current component (*i*). The reconstructed values *Ra*, *Rb*, *Rc*, and *Rd* used for context modelling and prediction correspond to the current component.

16. Return to Step 2. If all the samples of $\mathbf{V_i}$ consecutive lines of the i^{th} component have been processed, continue with samples of component $i + 1$ (or component 1, if i was the last component). Otherwise, continue with samples from component *i*.

17. The run and run interruption sample encoding procedures in Steps 17 and 18 of A.8 shall be followed, using the copy of *RUNindex* corresponding to the component *i*. The same test as in Step 16 above shall be performed to determine which component follows in the procedure.

B.3 Sample interleaved mode

B.3.1 Description

This mode is specified by setting the parameter **ILV** in the start of scan marker segment to a value of 2. In this mode, one sample at a time per component shall be encoded. The runs are common to all the components. The encoder shall only enter a run mode if the condition for run mode is satisfied for all the components in the scan, and shall continue in the run mode only if the condition for continuation is also satisfied for all the components. A single number, equal to the number of consecutive times the joint condition for continuation is met, shall be encoded with the procedure described in A.7, representing the length of the joint run. Only one variable *RUNindex* is used.

In the run interruption sample encoding procedure, the sample shall be encoded with *RItype*=0, as the decoder has no knowledge of the cause of the run interruption (the run stops when any of the components runs are interrupted). The same value of the variable *glimit* is used for all the components, corresponding to the common value of *RUNindex*. As in the line interleaved mode, the same counters shall be used across all components, and the prediction and context determination procedures shall be performed as in the single-component mode, and shall not use information from the multiple components.

NOTE – It is only in this mode that there are 365 possible regular contexts rather than 364. The additional context in this mode arises as a run mode is not necessarily implied when $Q1 = Q2 = Q3 = 0$ for a given component.

In this interleaved mode all components which belong to the same scan shall have the same dimensions.

B.3.2 Process flow

The process flow for the sample interleaved mode is described below in terms of the general process flow given in A.8. For convenience, the steps are numbered identically in this clause.

1. Initialise a single set of variables, as in step 1 in the procedure described in A.8.

2. Compute the local gradients for all the components as in Step 2 in A.8, in a component-independent form. At this step, three local gradients shall be computed for each component.

3. If all the local gradients for all the components are smaller than or equal to **NEAR** in absolute value, go into run mode, otherwise go to regular mode.

4. Follow steps 4 to 15 in the procedure described in A.8 for each one of the current samples of each component. Steps 4-15 for the sample *j* of the component *i* shall be completed before the steps 4-15 for the sample *j* of the next component *i*+1 are performed. Steps 4-15 of sample *j*+1 of any component are not performed until these steps are completed for all the samples *j* for all the components. The same set of variables is used in these steps, but the context determination and the prediction are performed for each component separately. The encoding of the sample *j* in component *i*+1 uses the variables already updated by the sample *j* in the previous component *i*.

16. All the samples in the same position *j*, for all the components, have now been encoded. The encoder shall now return to step 2 above to continue with the sample in position *j*+1 for all the components.

17. The run mode encoding procedure shall be followed.

    a) The condition in Step 17.a) in A.8 is tested for all the components, using *Ix* and *Ra* corresponding to the same component. The run continues if and only if the condition holds for all the components. If the run continues, then *Rx* shall be set to the corresponding value of *Ra* in the same component.

    b) Perform steps 17.b) to 17.f) in A.8. These steps are performed only once, since the runs are common for all the components, and one number represents the length of the joint run.

18. Perform the operations in A.7.2 (run interruption sample encoding), for each of the components. The state variable *RItype*=0 at all times, and the procedures in Code segment A.17 shall be skipped. Each run interruption sample shall be completely encoded before starting to process the next sample.

19. Return to Step 2.

B.4 Minimum Coded Unit (MCU)

For non-interleaved mode (**Ns** = 1, **ILV** = 0), the minimum coded unit is one line. For sample interleaved mode (**Ns** > 1, **ILV** = 2), the MCU is a set of **Ns** lines, one line per component, in the order specified in the scan header.

NOTE – The order in the scan header is determined by the order in the frame header.

For line interleaved mode (**Ns** > 1, **ILV** = 1), the MCU is $\mathbf{V_1}$ lines of component $\mathbf{C_1}$ followed by $\mathbf{V_2}$ lines of component $\mathbf{C_2}$ … followed by $\mathbf{V_{Ns}}$ lines of component $\mathbf{C_{Ns}}$. In addition, the encoding process shall extend the number of lines if necessary so that the last MCU is completed. Any line added by an encoding process to complete a last partial MCU shall be removed by the decoding process.

Annex C

Compressed data format

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

This annex specifies three compressed data formats for the JPEG-LS processes:

1. the interchange format;
2. the abbreviated format for compressed image data;
3. the abbreviated format for mapping tables and parameters specification data.

These compressed data formats closely follow the compressed data formats specified in Annex B of CCITT Rec. T.81 | ISO/IEC 10918-1. Markers identify the various structural parts of the compressed data format. Marker segments for two markers from the **JPG$_n$** markers reserved for JPEG extensions in CCITT Rec. T.81 | ISO/IEC 10918-1 are specified in this annex. The specifications for lossless processes in Annex B of CCITT Rec. T.81 | ISO/IEC 10918-1 apply unless explicitly revised in this annex.

NOTE – Implementers should obtain the latest version of Annex B of CCITT Rec. T.81 | ISO/IEC 10918-1 for normative referencing purposes.

C.1 General aspects of the compressed data format specification

In this annex, the terms "coding processes", "encoding process", "decoding process", and "compressed data" refer, respectively, to the lossless and near-lossless coding processes, encoding process, decoding process, and compressed data as defined in this Recommendation | International Standard.

C.1.1 Marker assignments

Two markers from the **JPG$_n$** set (which were reserved for JPEG extensions) are assigned in this Recommendation | International Standard, in addition to the marker code assignments in Table B.1 of CCITT Rec. T.81 | ISO/IEC 10918-1.

SOF$_{55}$ (X'FFF7') identifies a new Start of Frame marker for JPEG-LS processes.

LSE (X'FFF8') identifies marker segments used for JPEG-LS preset parameters.

Each of these markers starts a marker segment. These marker segments begin with a two-byte segment length parameter.

In addition, the **SOI**, **EOI**, **SOS**, **DNL**, **DRI**, **RST$_m$**, **APP$_n$**, and **COM** are valid markers in JPEG-LS.

All other markers defined in Annex B of CCITT Rec. T.81 | ISO/IEC 10918-1 shall not be present in JPEG-LS compressed data.

C.1.2 Coded data segments

A coded image data segment contains the output of the encoding process defined in this Recommendation | International Standard for a restart interval. It consists of an integer number of bytes.

NOTE – Making the coded image data segment an integer number of bytes is achieved as follows: 0-bits are used, if necessary, to pad the end of the coded image data to complete the final byte of a segment. In order to provide for easy detection of marker segments, any X'FF' byte generated by the encoding process defined in this Recommendation | International Standard is followed by a "stuffed" zero bit, provided that the X'FF' byte is not part of an inserted marker segment.

C.2 General JPEG-LS coding syntax

This clause specifies the interchange format syntax which applies to the JPEG-LS coding process.

C.2.1 High level syntax

The high level syntax shown in Figure B.2 of CCITT Rec. T.81 | ISO/IEC 10918-1 applies to the JPEG-LS coding processes defined in this Recommendation | International Standard.

C.2.2 Frame header syntax

The new frame marker **SOF$_{55}$** (X'FFF7') is defined for JPEG-LS coding. The frame header specified in B.2.2 of CCITT Rec. T.81 | ISO/IEC 10918-1 applies with the following differences:

– A **Y** (number of lines) value of zero means either that the value is defined in an **LSE** marker segment (which could precede or follow the **SOF$_{55}$** but shall not occur later than immediately following the first scan) or in a **DNL** marker immediately following the first scan. **Y** shall not be changed after that point.

– An **X** (number of columns) value of zero is allowed and means that the value shall be defined in an **LSE** marker segment. The value shall be defined before the first **SOS** marker and shall not be changed after the first scan.

A non-zero value of **Y** or of **X** shall not be changed by **LSE** marker segments.

C.2.3 Scan header syntax

The scan header specified in clause B.2.3 of CCITT Rec. T.81 | ISO/IEC 10918-1 applies to JPEG-LS coding with the following differences.

– The start of spectral selection (Ss) parameter is replaced with the **NEAR** parameter.

– The end of spectral selection (Se) parameter is replaced with the **ILV** parameter.

– For lossless coding, the **NEAR** parameter shall be zero. For near-lossless coding, this parameter is expressed as a positive quantity from 1 to $\min\left(255, \frac{\mathbf{MAXVAL}}{2}\right)$ (see C.2.4.1.1 for a definition of **MAXVAL**).

– For a single component scan, the **ILV** parameter is specified as having the value 0. In a line interleaved scan, **ILV** is specified as having the value 1. In a sample interleaved scan, **ILV** is specified as having the value 2. Sample interleave is allowed in a scan only if the components of the scan have an identical number of columns and of lines. The interleave modes are defined in Annex B.

– The $\mathbf{Td_i Ta_i}$ (the DC and AC entropy table selectors) byte is replaced with a mapping table selector $\mathbf{Tm_i}$ byte. The table selected with $\mathbf{Tm_i}$ shall have been specified by the time the decoder is ready to decode the scan containing component $\mathbf{C_i}$. Mapping table selectors can have values from 0 to 255 (see C.2.4.1.2). A value of zero indicates that no mapping table will be used for that component. If the point transform (**Pt**), specified by the **Al** parameter (see CCITT Rec. T.81 | ISO/IEC 10918-1), is not zero, all $\mathbf{Tm_i}$ shall be zero.

C.2.4 Table-specification and miscellaneous marker segment syntax

This Recommendation | International Standard requires an additional marker segment, **LSE**, which is described below.

C.2.4.1 JPEG-LS preset parameters specification syntax

The **LSE** marker segment may be present where tables or miscellaneous marker segments may appear. If tables specified in this marker segment for a given table ID appear more than once, each specification shall replace the previous specification. Figure C.1 specifies the marker segment following an **LSE** marker which defines either the JPEG-LS preset coding parameters whose values are used to override default parameter values, or mapping tables, or oversize image dimensions, and are collectively called "JPEG-LS preset parameters". Each specification of a JPEG-LS preset parameter in an **LSE** marker segment shall replace the previous specification.

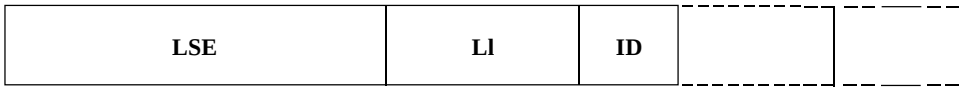


Figure C.1 – JPEG-LS preset parameters marker segment syntax

The marker and parameters shown in Figure C.1 are defined below.

LSE JPEG-LS preset parameters marker; marks the beginning of the JPEG-LS preset parameters marker segment.

Ll JPEG-LS preset parameters length; specifies the length of the JPEG-LS preset parameters marker segment shown in Figure C.1.

ID parameter ID; specifies which JPEG-LS preset parameters follow. If **ID** = X'01', the JPEG-LS preset coding parameters follow. If **ID** = X'02', a mapping table specification follows. If **ID** = X'03', a mapping table continuation follows. If **ID** = X'04', **X** and **Y** parameters greater than 16 bits are defined.

NOTE – The value of the length field in the definition of **Ll** and in subsequent descriptions of marker segments does not include the marker (see B.1.1.4 of CCITT Rec. T.81 | ISO/IEC 10918-1).

The parameters for each specified ID are specified in the following subclauses.

C.2.4.1.1 JPEG-LS preset coding parameters

Figure C.2 specifies the JPEG-LS preset coding parameters which follow the **LSE** when **ID** is equal to X'01'.

LSE	LI	1	MAXVAL	T1	T2	T3	RESET

Figure C.2 – LSE marker segment for JPEG-LS preset coding parameters

The new parameters shown in Figure C.2 are defined below. The size and allowed values of each parameter are given in Table C.1.

MAXVAL maximum possible value for any image sample in the scan. This must be greater than or equal to the actual maximum value for the components in the scan.

T1 first quantization threshold value for the local gradients.

T2 second quantization threshold value for the local gradients.

T3 third quantization threshold value for the local gradients.

RESET value at which the counters *A*, *B*, and *N* are halved.

Table C.1 – LSE marker segment parameter sizes and values for JPEG-LS preset coding parameters

Parameter	Size (bits)	Values
Ll	16	13
ID	8	1
MAXVAL	16	0, or $1 \le$ **MAXVAL** $< 2^{\mathbf{P}}$
T1	16	0, or **NEAR** $+ 1 \le$ **T1** $\le$ **MAXVAL**
T2	16	0, or **T1** $\le$ **T2** $\le$ **MAXVAL**
T3	16	0, or **T2** $\le$ **T3** $\le$ **MAXVAL**
RESET	16	0, or $3 \le$ **RESET** $\le$ max(255, **MAXVAL**)

NOTE 1 – **P** is the number of bits per image sample, contained in the start of frame marker segment, as defined in CCITT Rec. T.81 | ISO/IEC 10918-1**.**

For **MAXVAL, T1, T2, T3** and **RESET**, a value of 0 indicates reverting to default values as given in Table C.2. The SOI marker also resets these parameters to default values given in Table C.2. The default values of **T1**, **T2**, and **T3** specified in C.2.4.1.1.1 are calculated from the values of **MAXVAL** and **NEAR** that are in force at the time that **T1**, **T2**, and **T3** are used.

NOTE 2 – The values of **T1**, **T2**, and **T3** may change during the encoding process other than by an **LSE** marker segment, for example, if default values are used and the value of **NEAR** is changed by a scan header.

Table C.2 – Default values for JPEG-LS preset coding parameters

MAXVAL	$2^{\mathbf{P}}-1$
T1	See C.2.4.1.1.1
T2	See C.2.4.1.1.1
T3	See C.2.4.1.1.1
RESET	64

NOTE 3 – When an **LSE** marker is used, default parameter values can be specified by either using the value 0 or by explicitly specifying the default value in the marker segment.

C.2.4.1.1.1 Default threshold values

The default threshold values **T1, T2**, and **T3**, for gradient quantization, are given in terms of **MAXVAL, NEAR** and the "basic" default threshold values for the case **MAXVAL** = 255, lossless coding (**NEAR** = 0), denoted **BASIC_T1**, **BASIC_T2**, and **BASIC_T3**. These default values are given in Table C.3.

Table C.3 – Default threshold values in case MAXVAL = 255, NEAR = 0

BASIC_T1	3
BASIC_T2	7
BASIC_T3	21

The clamping function defined in Figure C.3 for integers *i* and *j* is also needed.

$$\text{CLAMP}(i, j) = \begin{cases} j & \text{if } i > \textbf{MAXVAL} \text{ or } i < j, \\ i & \text{otherwise.} \end{cases}$$

Figure C.3 – Clamping functions for default thresholds

In the case where **MAXVAL** ≥ 128, the dependence of the default values on **MAXVAL** is specified by **FACTOR** = $\lfloor$(min(**MAXVAL**, 4095) + 128)/256 . The default values in this case are given in Figure C.4.

T1 = CLAMP(**FACTOR** * (**BASIC_T1 –** 2) + 2 + 3***NEAR**, **NEAR** + 1)
T2 = CLAMP(**FACTOR** * (**BASIC_T2 –** 3) + 3 + 5***NEAR**, **T1**)
T3 = CLAMP(**FACTOR** * (**BASIC_T3 –** 4) + 4 + 7***NEAR**, **T2**)

Figure C.4 – Default values in case MAXVAL ≥ 128

Otherwise, if **MAXVAL** < 128, the dependence of the default values on **MAXVAL** is specified by **FACTOR** = $\lfloor$256/(**MAXVAL** + 1) . The default values in this case are given in Figure C.5.

T1 = CLAMP(max(2, $\lfloor$**BASIC_T1/FACTOR** + 3***NEAR**), **NEAR** + 1)
T2 = CLAMP(max(3, $\lfloor$**BASIC_T2/FACTOR** + 5***NEAR**), **T1**)
T3 = CLAMP(max(4, $\lfloor$**BASIC_T3/FACTOR** + 7***NEAR**), **T2**)

Figure C.5 – Default values in case MAXVAL < 128

C.2.4.1.2 Mapping table specification

Figure C.6 specifies the mapping table contained in the **LSE** marker segment when **ID** is equal to X'02'.

LSE	**LI**	**2**	**TID**	**Wt**	**Table [0..MAXTAB]**

Figure C.6 – LSE marker segment for mapping table specification

The new parameters shown in Figure C.6 are defined below. The size and allowed values of each parameter are given in Table C.4.

TID table ID; specifies the identification number of the table specified.

Wt width of table entries in bytes; specifies the number of bytes per entry in the selected table.

TABLE[0..MAXTAB] sample mapping table; each entry has **Wt** bytes.

MAXTAB defined in Figure C.7:

$$\mathbf{MAXTAB} = \begin{cases} \mathbf{MAXVAL} & \text{if } (5 + \mathbf{Wt} * (\mathbf{MAXVAL} + 1)) < 65535 \\ \left| \frac{65530}{\mathbf{Wt}} \right| - 1 & \text{otherwise} \end{cases}$$

Figure C.7 – Definition of MAXTAB

The value of **MAXVAL** in Figure C.7 is the one in force at the time the table is referred to in a scan header.

Table C.4 – LSE marker segment parameter sizes and values for mapping table specification

Parameter	Size (bits)	Values
Ll	16	5 + **Wt*(MAXTAB** + 1)
ID	8	2
TID	8	1 to 255
Wt	8	1 to 255
TABLE[*i*], *i* = 0..**MAXTAB**	**Wt***8	0 to $2^{\mathbf{Wt}*8} - 1$

When mapping tables are used, the decoder uses the reconstructed value *Rx* to index the corresponding table. If **MAXTAB** = **MAXVAL**, then the table specified in the **LSE** marker segment has one entry for each possible value *Rx*. Therefore, the table has **MAXVAL** + 1 entries, and the entries are written in the bit stream in ascending order of *Rx*. Each entry in the table contains **Wt** bytes. The decoder translates the value *Rx* to the **Wt**-byte long value **TABLE**[*Rx*]. This Recommendation | International Standard does not define an interpretation for the **Wt** bytes in an entry.

NOTE – A possible interpretation is a vector in some colour space. For example, with **Wt** = 3, the 3 bytes in a table entry could be interpreted as R,G,B values in a colour palette.

If **MAXTAB** < **MAXVAL**, then a complete mapping table with **MAXVAL** entries does not fit an **LSE** marker segment of maximum possible length (**Ll** = 65535 bytes), and the **LSE** marker segment with **ID** equal to X'02' must be immediately followed by one or more **LSE** marker segments with **ID** equal to X'03' ("mapping table continuation"), until a total of **MAXVAL** + 1 table entries have been specified.

C.2.4.1.3 Mapping table continuation

The structure of the mapping table continuation segment is similar to that of the preceding mapping table specification as specified in C.2.4.1.2, with the following differences:

- the length field **Ll** contains the number 5 + **Wt***(**MAXTABX** + 1), for a value **MAXTABX** defined in Figure C.8
- the **ID** field contains the value X'03'
- the table entries are **TABLE**[0..**MAXTABX**]

MAXTABX is defined in Figure C.8 in terms of the number **ENTRIES** of mapping table entries that are still unspecified following the most recent table mapping specification segment (with **ID** = X'02') and any associated mapping table continuation segments (with **ID** = X'03') preceding the current one.

$$\mathbf{MAXTABX} = \begin{cases} \mathbf{ENTRIES} - 1 & \text{if } (5 + \mathbf{Wt} * (\mathbf{ENTRIES} + 1)) < 65536 \\ \left\lfloor \dfrac{65530}{\mathbf{Wt}} \right\rfloor - 1 & \text{otherwise} \end{cases}$$

Figure C.8 – Definition of MAXTABX for mapping table continuation

The values **TID** and **Wt** shall be identical to those of the preceding mapping table specification segment. A mapping table continuation segment shall follow a mapping table specification segment with the same **TID** value, or another mapping table continuation segment with the same **TID** value.

If **MAXTABX** = **ENTRIES** – 1, then the current mapping table continuation segment is the last segment associated with the current **TID** value. If **MAXTABX** < **ENTRIES** – 1, then **ENTRIES** is reduced by (**MAXTABX** + 1), and is followed by more mapping table continuation segments with the same **TID** value.

Table entries shall be written in increasing order of *Rx*, within a mapping table specification or continuation marker segment, and from one segment to the next.

LSE marker segments may be present anywhere in the compressed image data where tables or miscellaneous marker segments are permitted. At the time the table is referred to, the number of its entries (as determined by the mapping table specification segment and any associated mapping table continuation segments) must be consistent with the value of **MAXVAL** currently in effect.

C.2.4.1.4 Oversize image dimension

Figure C.9 specifies the oversize image dimension parameters contained in the **LSE** marker segment when **ID** is equal to X'04'. The oversize image dimension parameters enable the specification of image dimensions **Ye** and **Xe** that can be larger than $2^{16} - 1$.

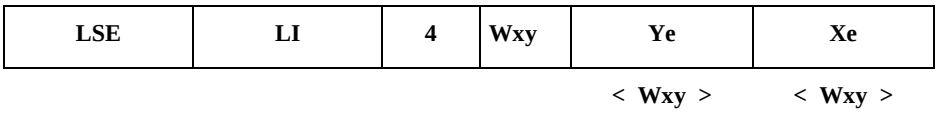


Figure C.9 – LSE marker segment for oversize image dimension

The new parameters shown in Figure C.9 are defined below. The size and allowed values of each parameter are given in Table C.5.

Wxy number of bytes used to represent **Ye** and **Xe**.

Ye number of lines in the image.

Xe number of columns in the image.

Table C.5 – LSE marker segment parameter sizes and values for oversize image dimension

Parameter	Size (bits)	Values
Ll	16	4+2*__Wxy__
ID	8	4
Wxy	8	2 to 4
Ye	**Wxy***8	0 to $2^{\mathbf{Wxy*8}}-1$
Xe	**Wxy***8	1 to $2^{\mathbf{Wxy*8}}-1$

A **Ye** value of zero means that the value is defined either in a **DNL** marker immediately following the first scan or in a following **LSE** marker segment that shall not occur later than immediately following the first scan.

C.2.5 Restart interval definition syntax

The restart interval marker segment is specified in Figure B.9 of CCITT Rec. T.81 | ISO/IEC 10918-1. Table B.7 of CCITT Rec. T.81 | ISO/IEC 10918-1 is modified in this Recommendation | International Standard to allow the segment length to vary from 4 to 6 bytes. This permits the restart interval to vary from two to four bytes to accommodate the largest possible number of columns and lines. If the restart interval is a 24- or 32-bit parameter, the convention still applies that the Most Significant Bit (MSB) shall come first and the Least Significant Bit (LSB) shall come last.

C.2.6 Define number of lines syntax

Figure B.12 of CCITT Rec. T.81 | ISO/IEC 10918-1 specifies the marker segment which defines the number of lines. Table B.10 of CCITT Rec. T.81 | ISO/IEC 10918-1 is modified in this Recommendation | International Standard to allow the segment length to vary from 4 to 6 bytes. This permits the number of lines to vary from two to four bytes to accommodate the larger possible number of lines. If the number of lines parameter is a 24- or 32-bit parameter, the convention still applies that the MSB shall come first and the LSB shall come last.

C.3 Abbreviated format for compressed image data

LSE marker segments which define mapping tables may be omitted if the application environment provides methods for table specification other than by means of the compressed image data.

C.4 Abbreviated format for table-specification data

LSE marker segments which define mapping tables may be present in compressed data that have no frames.

Annex D

Control procedures

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

This annex describes the encoder control procedures for the encoding process.

NOTE 1 – There is **no requirement** in this Recommendation | International Standard that any encoder or decoder shall implement the procedures in precisely the manner specified by the flow charts in this annex. It is necessary only that an encoder or decoder implement the **function** specified in this annex. The sole criterion for an encoder or decoder to be considered in conformance with this Recommendation | International Standard is that it satisfy the requirements given in clause 8.

NOTE 2 – Implementation-specific setup steps are not indicated in this annex and may be necessary.

D.1 Control procedure for encoding an image

The encoder control procedure for encoding an image is shown in Figure D.1.

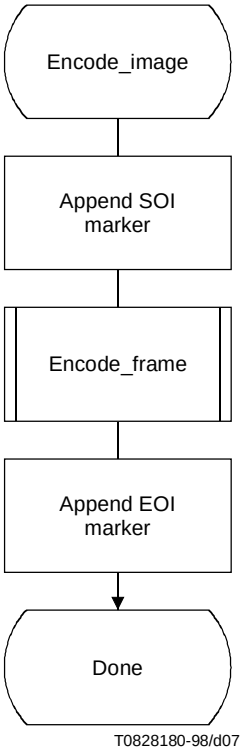


Figure D.1 – Control procedure for encoding an image

D.2 Control procedure for encoding a frame

In all cases where markers are appended to the compressed image data, optional X'FF' fill bytes may precede the marker.

The control procedure for encoding a frame is oriented around the scans in the frame. The frame header is first appended, and then the scans are coded. Table specifications and other marker segments may precede the $\mathbf{SOF_{55}}$ marker, as indicated by [Append tables/miscellaneous] in Figure D.2.

Figure D.2 shows the encoding process frame control procedure.

D.3 Control procedure for encoding a scan

A scan consists of a single pass through the data of each component in the scan. Table specifications and other marker segments may precede the **SOS** marker. If more than one component is coded in the scan, the data are interleaved. If restart is enabled, the data are segmented into restart intervals. If restart is enabled, a $\mathbf{RST_m}$ marker is placed in the coded data between restart intervals. If restart is disabled, the control procedure is the same, except that the entire scan contains

a single restart interval. The compressed image data generated by a scan are always followed by a marker, either the **EOI** marker or the marker of the next marker segment.

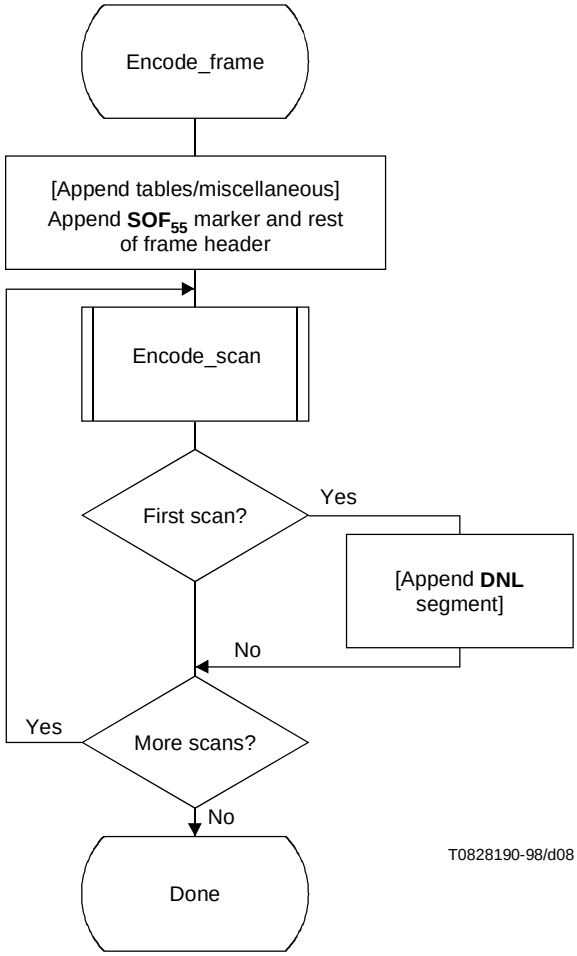


Figure D.2 – Control procedure for encoding a frame

Figure D.3 shows the encoding process scan control procedure. The loop is terminated when the encoding process has coded the number of restart intervals which make up the scan. "*m*" is the restart interval modulo counter needed for the **RST$_m$** marker. The modulo arithmetic for this counter is shown after the "Append **RST$_m$** marker" procedure.

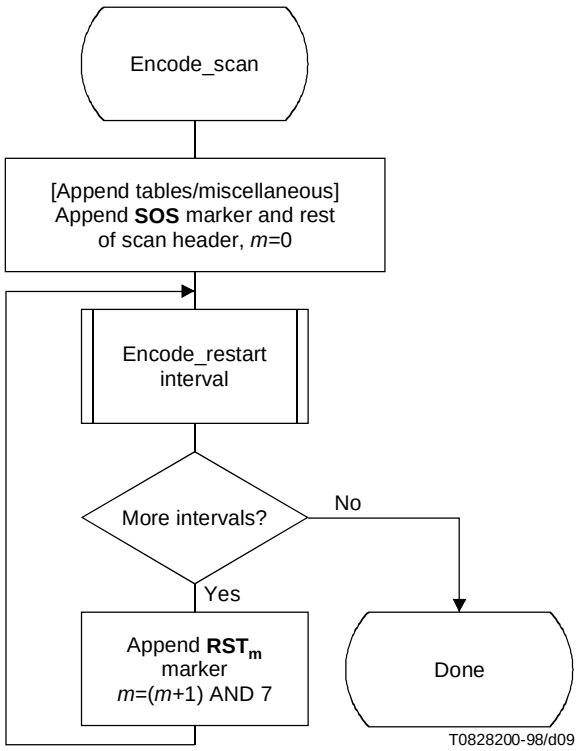


Figure D.3 – Control procedure for encoding a scan

D.4 Control procedure for encoding a restart interval

Figure D.4 shows the encoding process control procedure for a restart interval. The loop is terminated either when the encoding process has coded the number of minimum coded units (MCU) in the restart interval or when it has completed the image scan.

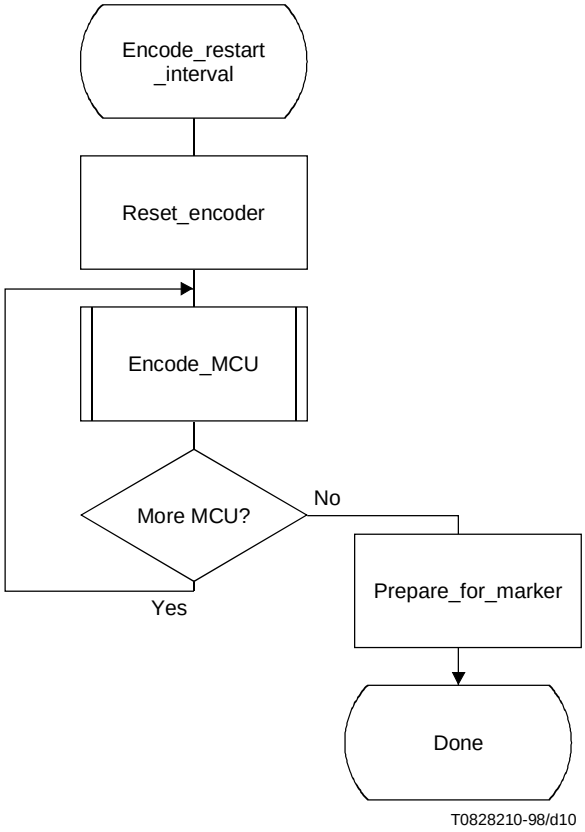


Figure D.4 – Control procedure for encoding a restart interval

The "Reset_encoder" procedure consists at least of the following:

a) initialise variables according to the corresponding interleave mode as if the first line of each component in the restart interval were the first line of the same component in a scan (see A.2.1, B.2.2 and B.3.2);

b) do all other implementation-dependent setups that may be necessary.

The procedure "Prepare_for_marker" terminates the coded image data segment by:

a) padding the final byte with zero bits

NOTE – The number of minimum coded units (MCU) in the final restart interval must be adjusted to match the number of MCUs in the scan. The number of MCUs is calculated from the frame and scan parameters.

D.5 Control procedure for encoding a Minimum Coded Unit (MCU)

The minimum coded unit is defined in Annex B. Within a given MCU the samples are coded in the order in which they occur in the MCU. The control procedure for encoding an MCU is shown in Figure D.5.

In Figure D.5, **Nb** refers to the number of samples in the MCU. The order in which samples occur in the MCU is defined in Annex B.

The procedures for encoding a sample are specified in Annexes A and B.

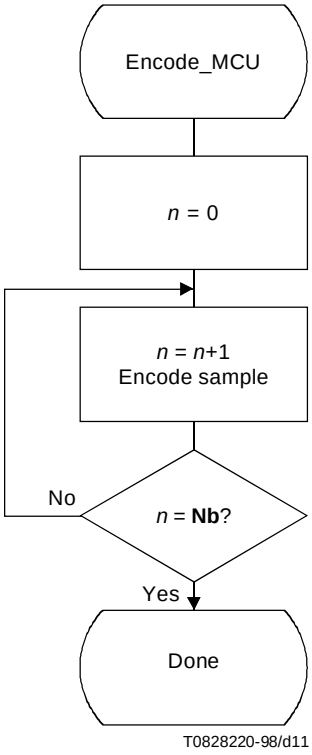


Figure D.5 – Control procedure for encoding a Minimum Coded Unit (MCU)

Annex E

Conformance tests

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

This annex specifies test images for conformance testing as specified in clause 8.

E.1 Test images

The conformance tests for encoders and decoders specified in this Recommendation | International Standard are based on a collection of test images, which form part of this Recommendation | International Standard, and consist of a set of single- and multi-component image data, both source image data, and image data encoded in accordance with the processes specified in this Recommendation | International Standard.

Source images are stored as computer files in PGM (single-component) or PPM (multi-component) format. Details of these formats are given in E.1.3. Components of a three-component multi-component test image are labelled "red", "green", and "blue". Compressed image data is stored in digital computer files in the format specified in this Recommendation | International Standard.

E.1.1 Source images

The list of source test images is detailed in Table E.1.

Table E.1 – Source test images

Image name	Precision (bits per sample)	Components	Dimensions (columns × lines)	Comments
TEST8	8	3	256 ×256	Reference 8-bps colour image
TEST8R	8	1	256 × 256	"red" component of TEST8
TEST8G	8	1	256 × 256	"green" component of TEST8
TEST8B	8	1	256 × 256	"blue" component of TEST8
TEST8GR4	8	1	256 × 64	"green" component of TEST8, sub-sampled 4X in the vertical direction
TEST8BS2	8	1	128 × 128	"blue" component of TEST8, sub-sampled 2X in both vertical and horizontal directions
TEST16	12	1	256 × 256	Reference 12-bps monochrome image

TEST8 is an 8-bit per sample RGB colour image composed of areas of photographic, graphic, text, and random data. Other images starting with the prefix TEST8 are derived from TEST8 as indicated in the "Comments" column of Table E.1. TEST16 is a 12-bit per sample monochrome image with a similar composition. Sub-sampling an image component by mX is achieved by using every m-th sample of the component in the appropriate direction, starting from the first sample, and without interpolation.

NOTE – The mixture of data in the test images was designed to exercise many paths of the encoding and decoding processes. There is no guarantee, however, that every possible path of the processes will be exercised.

E.1.2 Compressed image data

The list of compressed image data is detailed in Table E.2.

Each compressed image contains one frame, with the number of scans specified in the table. For multi-scan images, each scan contains one component ("red", "green" and "blue" in this order), and the same **NEAR** parameter is used for all scans. Scans contain no restart markers.

For Test No. 7 and Test No. 8, the components of image TEST8 are given in the "Source Image(s)" column and are sub-sampled as indicated in Table E.1.

For parameters not explicitly specified in the table, all tests use default values as defined in Annex C, except for Tests Nos. 9 and 10, which use non-default values for **T1, T2, T3** and **RESET**.

Table E.2 – Compressed image data

Test No.	Compressed file name	Source image(s)	Compo-nents	NEAR	Scans	ILV	Sub-sampling	Other JPEG-LS parameters
1	T8C0E0.JLS	TEST8	3	0	3	0	none	default
2	T8C1E0.JLS	TEST8	3	0	1	1 (line)	none	default
3	T8C2E0.JLS	TEST8	3	0	1	2 (sample)	none	default
4	T8C0E3.JLS	TEST8	3	3	3	0	none	default
5	T8C1E3.JLS	TEST8	3	3	1	1 (line)	none	default
6	T8C2E3.JLS	TEST8	3	3	1	2 (sample)	none	default
7	T8SSE0.JLS	TEST8R TEST8GR4 TEST8BS2	3	0	1	1 (line)	See below	default
8	T8SSE3.JLS	TEST8R TEST8GR4 TEST8BS2	3	3	1	1 (line)	See below	default
9	T8NDE0.JLS	TEST8BS2	1	0	1	0	none	**T1**=**T2**=**T3**=9 **RESET**=31
10	T8NDE3.JLS	TEST8BS2	1	3	1	0	none	**T1**=**T2**=**T3**=9 **RESET**=31
11	T16E0.JLS	TEST16	1	0	1	0	none	default
12	T16E3.JLS	TEST16	1	3	1	0	none	default

E.1.3 Test image formats

For the purpose of conformance testing, the source test images are stored in computer files using the following formats. All character coding in the formats is in accordance with ISO/IEC 646:1991.

NOTE – These formats are defined only for the purpose of distributing test images for conformance testing as part of this Recommendation | International Standard. This Recommendation | International Standard does not prescribe any specific format as input for the encoding process, or as output from the decoding process.

E.1.3.1 PGM format (for single-component images)

The file starts with a header consisting of 3 lines in the following format:

P5
X Y
MAXVAL

Here, "P5" is text coded in accordance with ISO/IEC 646, **X** and **Y** are, respectively, the number of columns and lines of the image (decimal integers represented in character coded format, separated by a space), and **MAXVAL** is the maximum sample value (a decimal integer represented in character coded format). As an example, the header for TEST16.PGM has the following format:

P5
256 256
4095

The header is followed by **X*****Y** samples in binary format, stored in raster scan order, line by line. For TEST8 and its derived images, each sample occupies one byte. For TEST16, each sample occupies two bytes, with the most significant byte of the sample stored before the least significant byte, and with the 12 bits of the sample stored in the least significant bits of the two bytes representing the sample.

E.1.3.2 PPM format (for multi-component images)

This is a three-component file format which starts with a header consisting of three lines in the following format:

P6
X Y
MAXVAL

Here, "P6" is character coded text, **X** and **Y** are, respectively, the number of columns and lines of the image (decimal integers represented in character coded format, separated by a space), and **MAXVAL** is the maximum sample value (a decimal integer represented in character coded format). As an example, the header for TEST8.PPM has the following format:

P6
256 256
255

The header is followed by 3***X*****Y** samples in binary format, stored in raster scan order, line by line, and column by column, with samples interleaved from each component in "red", "green", "blue" order. For TEST8 and its derived images, each sample occupies one byte (thus, each sample position within a line occupies 3 bytes, corresponding to the 3 image components).

Annex F

Decoding procedures

(This annex does not form an integral part of this Recommendation | International Standard)

F.1 Process flow

The coding specified in this Recommendation | International Standard is fairly symmetric, meaning that both the encoding and decoding processes use the same basic procedures and follow almost the same steps in reverse order (besides a few sign changes). For the decoding process, therefore, only the process flow is shown. Details on the different procedures can be found in the detailed clauses in Annex A.

NOTE – There is **no requirement** in this Recommendation | International Standard that any encoder or decoder shall implement the procedures in precisely the manner specified in this Annex. It is necessary only that an encoder or decoder implement the **function** specified in this Annex. The sole criterion for an encoder or a decoder to be considered in conformance with this Recommendation | Standard is that it satisfy the requirements determined by the conformance tests given in clause 8.

1. Initialisation:
   a) Assign default values to non-specified JPEG-LS preset coding parameters (see A.1).
   b) Initialise the non-defined samples of the causal template (see A.2.1).
   c) Compute the variable **RANGE** (see A.2.1). For lossless coding, **RANGE** = **MAXVAL** + 1. For near-lossless coding, $\mathbf{RANGE} = \left\lfloor \frac{\mathbf{MAXVAL} + 2 * \mathbf{NEAR}}{2 * \mathbf{NEAR} + 1} \right\rfloor + 1$

      Compute the parameters **qbpp** = $\lceil \log \mathbf{RANGE} \rceil$, **bpp** = $\max(2, \lceil \log(\mathbf{MAXVAL} + 1) \rceil)$, and **LIMIT** = 2*(**bpp** + max(8, **bpp**)).
   d) For each context Q, initialise four variables (see A.2.1): $A[Q] = max\left(2, \left\lfloor \frac{\mathbf{RANGE} + 2^5}{2^6} \right\rfloor\right)$, $B[Q] = C[Q] = 0$, $N[Q] = 1$. For $A[Q]$ and $N[Q]$, Q is an integer between 0 and 366; for $B[Q]$ and $C[Q]$, Q is an integer between 0 and 364 (regular mode contexts only).
   e) Initialise the variables for the run mode: $RUNindex = 0$ and **J**[0..31] = {0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 9,10,11,12,13,14,15}.
   f) Initialise the two run interruption variables $Nn[365]$, $Nn[366] = 0$.
2. Compute the local gradients according to Code segment A.1.
3. Select the mode following the procedure in Code segment A.2. If run mode is selected, go to run mode (step 18), otherwise continue with the regular mode.
4. Quantize the local gradients according to the steps detailed in Code segment A.4.
5. Check and change if necessary the sign of the context, modifying accordingly the variable *SIGN* (see A.3.4).
6. Compute *Px* according to Code segment A.5.
7. Correct *Px* using $C[Q]$ and the variable *SIGN* and clamp the corrected value to the interval [0..**MAXVAL**] according to the procedure in Code segment A.6.
8. Compute the context-dependent Golomb variable k according to the procedure in Code segment A.10.
9. Decode the mapped error value *MErrval*:
   a) Read the unary code. If it contains less than **LIMIT** – **qbpp** – 1 zeros, use it to form the most significant bits of *MErrval* and read k additional bits, to compose the k least significant bits of *MErrval*.
   b) If the unary code contains **LIMIT** – **qbpp** – 1 zeros, read **qbpp** additional bits to get a binary representation of *MErrval* –1.
10. Perform the inverse of the error mapping indicated in Code segment A.11, where now *MErrval* is given and *Errval* is computed.
11. Update the variables according to Code segment A.12.

12. For near-lossless coding, multiply *Errval* by (2***NEAR**+1).

13. Invert sign of *Errval* if the variable *SIGN* is negative.

14. Compute $Rx = (Errval + Px)$ modulo [**RANGE***(2***NEAR**+1)]. For near-lossless coding, map *Rx* to the interval [–**NEAR**..**RANGE***(2***NEAR**+1)–1–**NEAR**]. Clamp *Rx* to [0..**MAXVAL**]. This is done by the following procedure, where the C programming language is used as specified in A.2.2:

```
if (Rx < –NEAR)
    Rx = Rx + RANGE * (2 * NEAR + 1);
else if (Rx > MAXVAL + NEAR)
    Rx = Rx – RANGE * (2 * NEAR + );

if (Rx < 0)
    Rx = 0;
else if (Rx > MAXVAL)
        Rx = MAXVAL;
```

15. Map *Rx* using the inverse point transform **Pt** specified by the parameter **Al** (see B.2.3 of CCITT Rec. T.81 | ISO/IEC 10918-1) and the applicable mapping table, if any, as specified in Annex C.

16. Compute the prediction correction value *C*[*Q*] according to the procedure in Code segment A.13.

17. Process next sample of the same component or of next component, starting from Step 2, as specified in Annex B.

18. Run mode decoding:

    a) Read a bit, R, from the bit stream.

    b) If R = '1' then

        i) Fill the image with $2^{\mathbf{J}[RUNindex]}$ samples of value *Ra*, or until a line is completed.

        ii) If exactly $2^{\mathbf{J}[RUNindex]}$ samples were filled in the previous step, and *RUNindex*<31, then increase *RUNindex* by one. If the last sample in the line has not yet been decoded, return to step 18.a) to read more bits from the bit stream. Otherwise, go to Step 17.

    c) If R = '0' then

        i) Read **J**[*RUNindex*] bits from the bit stream and fill the image with the value Ra for as many samples as the number formed by these bits (MSB first).

        ii) If *RUNindex*>0, decrement *RUNindex* by one.

        iii) Decode the run interruption sample value, reversing the procedures in A.7.2.

        iv) Go to Step 17.

Annex G

Description of the coding process

(This annex does not form an integral part of this Recommendation | International Standard)

G.1 Context modelling

G.1.1 Derivation of gradient

The context modelling procedure specified in Annex A uses the causal template *a*, *b*, *c*, and *d* depicted in Figure 3. The context that conditions the coding of the current sample is built from the differences *D1*, *D2*, and *D3* of the reconstructed values *Ra*, *Rb*, *Rc*, and *Rd* at the sample positions *a*, *b*, *c*, and *d: D1* = Rd–Rb, *D2* = *Rb–Rc*, and *D3* = *Rc–Ra*. In lossless coding, the reconstructed values are identical to those of the source image data. These differences are referred to as the local gradient, and capture the level of activity (smoothness, edginess) surrounding the sample at position *x*. These local gradient values are used to estimate the statistical behaviour of the prediction errors to be encoded.

Without further processing of the local gradient values *D1*, *D2*, and *D3*, a very large number of contexts could be generated. This has a number of disadvantages:

- If a small number of samples are coded in the same context, there will in general not be enough information to collect the relevant statistics for the context;
- The memory requirements to implement the coding procedures specified in this Recommendation | International Standard increase with the number of contexts.

Gradients with similar characteristics are therefore merged to create conditioning contexts. In this Recommendation | International Standard, only a small number of statistical parameters need to be estimated per context. This allows for a large number of contexts without excessive penalty in code length due to the number of parameters modelled (model cost), or in memory requirements.

G.1.2 Quantization

The gradient merging procedure is based on *quantizing* the local gradient defined above. Assuming the image to be symmetric (that is, there is no preference to vertical over horizontal orientations), *D1*, *D2*, and *D3* influence the modelling in the same way, and each of these differences is quantized into a small number of approximately equiprobable regions.

The probability of a local gradient taking the value *v* is assumed to be the same as the probability of it taking the value *–v*. The quantizer is therefore symmetric about a difference value of zero. A further reduction in the number of contexts is obtained by merging quantized gradients of opposite signs. The quantized triplet (*Q1*, *Q2*, *Q3*) is merged with the triplet (*–Q1*, *–Q2*, *–Q3*). This last merging procedure is compensated by changing the sign of the prediction error.

G.1.3 Prediction

G.1.3.1 Prediction basis

In the context modelling procedure, the local gradients are quantized. In order to partially compensate for this information loss, the context determination procedure is followed by a prediction step. The idea behind this procedure is that the value at the current sample *x* can be estimated from the reconstructed values of the samples surrounding it. Then, instead of coding the value itself, the prediction error is encoded.

The prediction procedure in this Recommendation | International Standard is based on the subset of samples at positions *a*, *b*, and *c* of the causal template depicted in Figure 3, where *x* denotes the position of the current sample to be encoded.

G.1.3.2 Edge detection

The first step in the prediction procedure defined by this Recommendation | International Standard is to perform a simple test to detect vertical or horizontal edges. If an edge is not detected, then the predicted value *Px*, at the sample position *x*, is *Ra+Rb–Rc*, as this would be the value at *x* if a plane is passed through the *a*, *b*, and *c* sample locations, with respective heights *Ra*, *Rb*, *Rc*, and the constraint is imposed that the current sample belongs to the same plane. This constraint expresses the expected smoothness of the image in the absence of edges. If a vertical edge is detected, the value at *b*, (*Rb*), is predicted. If an horizontal edge is detected, the value at *a*, (*Ra*), is predicted. This procedure is performed by the simple formula in Figure G.1.

$$Px = \begin{cases} \min(Ra, Rb) & \text{if } Rc \geq \max(Ra, Rb) \\ \max(Ra, Rb) & \text{if } Rc \leq \min(Ra, Rb) \\ Ra + Rb - Rc & \text{otherwise.} \end{cases}$$

Figure G.1 – Basic edge-detecting predictor

G.1.3.3 Prediction correction

Following the basic predictor, the predicted value is corrected by a bias-dependent term. The encoding procedure in this Recommendation | International Standard assumes the distribution of prediction errors to be two-sided geometric, symmetric, and centred between –1 and 0. In context-based models, systematic, context-dependent biases in the prediction errors are not uncommon. To alleviate the effect of systematic biases, this Recommendation | International Standard defines a bias correction procedure aimed at centring the distribution of prediction errors in the targeted interval. This procedure is based on the accumulated value of errors incurred so far for samples of the same context. After this procedure, the final corrected prediction error is computed. For near-lossless coding, this error is then quantized.

G.2 Encoding in the regular coding mode

After the context is determined, and if it does not take the encoding process into the run mode, the prediction, bias, and corrected prediction error are computed. The last step of the encoding process in this mode of operation is to encode this corrected prediction error. For this, a scheme derived from Golomb coding is used. This means that only two statistical parameters, representing the decay rate of the distribution and its bias, have to be estimated for each context. All possible error values are mapped into non-negative ones prior to encoding.

G.2.1 Code definition

Golomb coding was first introduced as a means for encoding series containing non-negative run lengths. For a positive integer parameter *g*, the Golomb code of order *g* encodes an integer *n* greater than or equal to 0 in two parts: a *unary* representation of the integer part of *n/g*, and a *binary* representation of *n modulo g*. Golomb codes are optimal for geometric probability distributions for non-negative integers. For every distribution of this form, there exists a value of the parameter *g* such that the code yields the shortest possible average code length over all uniquely decipherable codes for non-negative integers.

G.2.2 Power-of-2 case

The special case of Golomb codes where *g* is a *k*-power of 2 leads to very simple encoding/decoding procedures: the code for *n* consists of the number formed by the higher order bits of *n*, in unary representation, followed by the *k* least significant bits of *n*. This specific case is the one used in this Recommendation | International Standard and is denoted by *G*(*k*). In the following example, *k* stands for the Golomb variable, where *g* is equal to the *k-th* power of 2.

G.2.3 Example

This example applies for *G*(2). The value *n*=19 is assumed. The binary representation of 19 is 10011. The two (*k*=2) lowest significant bits are sent as they are (11). This corresponds to 3 = 19 modulo 4. The remaining higher significant bits, 100, represent the integer part of the quotient 19/4, i.e. the number 4. This number is sent in unary form as four zeros and a terminating one, 00001. Combining both parts, with the unary part first, the code for *n*=19, with *k*=2, is 0000111.

G.2.4 Limited length Golomb code

In practice, when encoding a bounded set of non-negative integers, it is desirable to limit the maximum length of a Golomb code word (which for *G*(0) is $j + 1$, where *j* denotes the maximal integer in the set, often a large number) to a number *glimit* of bits. A method for this is to use the code *LG*(*k*, *glimit*) defined in this clause, where it is assumed $k < \lceil \log j \rceil$ (otherwise the expanding code *G*(*k*) would be systematically outperformed by $G(\lceil \log j \rceil - 1)$.

To encode a non-negative integer n, the number q formed by the higher order bits of n is computed. If $q < glimit - \lceil \log j \rceil - 1$, the encoding process proceeds as for $G(k)$. Since $k < \lceil \log j \rceil$, the total code length after appending k bits is within the required limit. If $q \geq glimit - \lceil \log j \rceil - 1$, then $glimit - \lceil \log j \rceil - 1$ is encoded in unary representation (i.e. $glimit - \lceil \log j \rceil - 1$ zeros followed by a one), which acts as an "escape code," followed by an explicit binary representation of $n-1$, with $\lceil \log j \rceil$ bits, for a total of $glimit$ bits. If $glimit > \lceil \log j \rceil + 1$, $n = 0$ always satisfies the condition for regular Golomb encoding, so that the length limitation is applied only in cases where $n > 0$, and the binary code for $n-1$ is $\lceil \log j \rceil$ bits long as claimed.

G.2.5 Coding negative values

Golomb codes were originally designed for non-negative integer values. Prediction errors from the prediction procedure described in Annex A can also be negative, and hence their distribution is in general two-sided geometric and symmetric, rather than one-sided. One way of extending the above coding scheme to handle this situation is to map all possible error values into non-negative ones prior to encoding. This requires a good estimation of the centre of this two-sided distribution, which is closely related to the bias measurement described in Annex A. In this Recommendation | International Standard, the mapping described in Annex A and Annex F approximates the optimal solution for two-sided geometric distributions. Table G.1 shows an example of coding of prediction errors with this mapping, and the limited length Golomb code $LG(2,32)$, for 8-bit alphabets ($\lceil \log j \rceil = 8$, following the notation of G.2.4). In this example, the limitation does not apply for mapped values smaller than 92.

Table G.1 – Example coding of prediction errors

Prediction error	Mapped value	Code $LG(2,32)$
0	0	1 00
–1	1	1 01
1	2	1 10
–2	3	1 11
2	4	01 00
–3	5	01 01
3	6	01 10
–4	7	01 11
4	8	001 00
–5	9	001 01
5	10	001 10
–6	11	001 11
6	12	0001 00
–7	13	0001 01
7	14	0001 10
–8	15	0001 11
8	16	00001 00
–9	17	00001 01
9	18	00001 10
13s4 –10	19	00001 11
10	20	000001 00
–11	21	000001 01
11	22	000001 10
–12	23	000001 11
12	24	0000001 00
…	…	…
50	100	00000000000000000000 00001 01100011

G.2.6 Parameter determination

One of the crucial steps in schemes using Golomb coding is the determination of the optimal value of the code parameter *k*, the value yielding the shortest possible average code length for the mapped prediction errors. In this Recommendation | International Standard, the value of *k* is context-dependent and varies adaptively. The value of *k* for each context is updated each time a sample belonging to that context is encoded. The updated value is based on the accumulated sum of absolute values of prediction errors that occurred in the same context and is defined in Annex A.

G.3 Encoding in the run mode

In a pure Huffman coding process, at least one bit per sample is needed. In order to increase the compression in uniform image areas, a run mode coding procedure is added in this Recommendation | International Standard.

If the reconstructed values at sample positions *a*, *b*, *c*, and *d* are identical, or their absolute difference is less than or equal to the allowed error in near-lossless coding, the process enters the run mode. In this mode, the encoder scans the image, starting from the sample at position *x* until a sample is met which is not identical to the reconstructed value of the sample at *a* (or not nearly identical within the bounds set for near-lossless coding), or the end of the current sample line is encountered. The encoder encodes the length of the run and the sample immediately after the run ends (if the run was not terminated by reaching the end of the current line). The procedure defined in this Recommendation | International Standard for coding run lengths can be viewed as an extension of Golomb coding. In run mode, the coding process does not use prediction.

Annex H

Examples and guidelines

(This annex does not form an integral part of this Recommendation | International Standard)

H.1 Introduction

This annex includes a number of examples intended to indicate how the encoding process works, and how the resulting bit stream should be output. The examples are intended to indicate the coding principles only, as the very small image data set used will in practice result in data expansion rather than compression, particularly after marker segments and file format information is added to the output bit stream.

H.2 Example of how bits are output in the bit stream

Assume the encoder to be at the beginning of the coded image data segment and that the encoding process outputs the following binary codes:

101001	(length 6)
111	(length 3)
1100000001	(length 10)

These binary numbers are written with the most significant bit in the leftmost position. The output bit stream will contain the byte 10100111 followed by 11100000. The current incomplete byte will contain 001xxxxx. The most significant bit of the next binary code will fill the 4th most significant bit of the current incomplete byte. If there were no more output codes after the three listed above, the incomplete byte would be padded with zeros as 0010000 to terminate the coded image data segment.

H.3 Detailed coding example

This coding example is based on the sixteen-byte sample image shown in Figure H.1.

Index	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	90	74	74
2	0	68	50	43	205	205
3	68	64	145	145	145	145
4	64	100	145	145	145	

T0828230-98/d12

Figure H.1 – Example image data

The inner box represents the actual image, whilst the shaded area represents the implied values for *Rb*,*Rc* and *Rd*, when the sample *Ix* is in the first line, for *Ra* and *Rc* when the sample *Ix* is in the first column, and for *Rd* when the sample *Ix* is in the last column. Lossless coding and default parameters are assumed.

NOTE – To represent the output bit stream, when more than 5 bits of the same kind appear one after the other, they will be denoted as the count of the bits, followed by the bit value in word form. For example, 1010000001 will be represented as 101+6 zero bits+1.

Firstly, Line 1, Samples 1 through 3 are encoded:

Rc=0	*Rb*=0	*Rd*=0
Ra=0	*Ix*=0	

D1=D2=D3=0, so run mode is entered.

The parameter values before encoding are:

NOTE – In all the tables in this example, *Errval* (mod) indicates the value of *Errval* after modulo division.

RUNval	*RUNcnt*	*RUNindex*	*RItype*	*Errval* (mod)	*TEMP*	*k*	*map*	*EMErrval*	*Q*	*A*[*Q*]	*N*[*Q*]	*Nn*[*Q*]
0	2	0	1	90	4	2	0	179	366	4	1	0

and hence the output bits are 1 1 + 23 zero bits + 1 1 0 1 1 0 0 1 0

The parameter values after updating are:

RUNindex	*A*[*Q*]	*N*[*Q*]	*Nn*[*Q*]
1	93	2	0

Line 1, Sample 4 is now coded:

Rc=0	*Rb*=0	*Rd*=0
Ra=90	*Ix*=74	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
0	0	4	90	–1	16	16	2	32	4	0	0	1

and hence the output bits are 8 zero bits + 1 0 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
20	0	1	2

Line 2, Sample 1 is now encoded:

Rc=0	*Rb*=0	*Rd*=0
Ra=0	*Ix*=68	

*D1=D2=D3=*0, so run mode is entered.

The parameter values before encoding are:

RUNval	*RUNcnt*	*RUNindex*	*RItype*	*Errval* (mod)	*TEMP*	*k*	*map*	*EMErrval*	*Q*	*A*[*Q*]	*N*[*Q*]	*Nn*[*Q*]
0	0	1	1	68	94	6	0	135	366	93	2	0

and hence the output bits are 0 0 0 1 0 0 0 1 1 1

The parameter values after updating are:

RUNindex	*A*[*Q*]	*N*[*Q*]	*Nn*[*Q*]
0	160	3	0

Line 2, Sample 2 is now encoded:

Rc=0	*Rb*=0	*Rd*=90
Ra=68	*Ix*=50	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
4	0	–4	68	1	–18	–18	2	35	4	0	0	1

and hence the output bits are 8 zero bits + 1 1 1

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
22	–1	–1	2

Line 2, Sample 3 is now encoded.

Rc=0	*Rb*=90	*Rd*=74
Ra=50	*Ix*=43	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
3	–4	4	90	–1	47	47	2	94	4	0	0	1

And hence the output bits are 23 zero bits + 1 0 1 0 1 1 1 0 1

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
51	0	1	2

Line 2, Sample 4 is now encoded.

Rc=90	*Rb*=74	*Rd*=74
Ra=43	*Ix*=205	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
0	3	–4	43	–1	–162	94	2	188	4	0	0	1

And hence the output bits are 23 zero bits + 1 1 0 1 1 1 0 1 1

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
98	0	1	2

Line 3, Sample 1 is now encoded.

Rc=0	*Rb*=68	*Rd*=50
Ra=68	*Ix*=64	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
3	–4	4	67	–1	3	3	5	6	51	0	1	2

And hence the output bits are 1 0 0 1 1 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
54	0	2	3

Line 3, Sample 2 is now encoded.

Rc=68	*Rb*=50	*Rd*=43
Ra=64	*Ix*=145	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
3	3	–2	50	–1	–95	–95	2	189	4	0	0	1

And hence the output bits are 23 zero bits + 1 1 0 1 1 1 1 0 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
99	–1	–1	2

Line 3, Sample 3 is now encoded.

Rc=50	*Rb*=43	*Rd*=205
Ra=145	*Ix*=145	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
4	–3	–4	138	1	7	7	2	14	4	0	0	1

And hence the output bits are 0 0 0 1 1 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
11	0	1	2

Line 3, Sample 4 is now encoded.

Rc=43	*Rb*=205	*Rd*=205
Ra=145	*Ix*=145	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
0	4	–4	205	1	–60	–60	2	119	4	0	0	1

And hence the output bits are 23 zero bits + 1 0 1 1 1 0 1 1 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
64	–1	–1	2

Line 4, Sample 1 is now encoded.

Rc=68	*Rb*=64	*Rd*=145
Ra=64	*Ix*=100	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
4	–2	2	64	1	36	36	2	72	4	0	0	1

And hence the output bits are 18 zero bits + 1 0 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
40	0	1	2

Line 4, Sample 2 is now encoded.

Rc=64	*Rb*=145	*Rd*=145
Ra=100	*Ix*=145	

The parameter values before encoding are:

Q1	*Q2*	*Q3*	*Px*	*SIGN*	*Errval*	*Errval* (mod)	*k*	*MErrval*	*A*[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
0	4	–4	144	1	1	1	5	2	64	–1	–1	2

And hence the output bits are 1 0 0 0 1 0

The parameter values after updating are:

A[*Q*]	*B*[*Q*]	*C*[*Q*]	*N*[*Q*]
65	0	–1	3

Line 4, Samples 3 through 4 is now encoded.

Rc=145	*Rb*=145	*Rd*=145
Ra=145	*Rx*=145	

D1=D2=D3=0, so enter run mode

The parameter values before encoding are (N/A indicates that the value is non-applicable, as no run interruption sample is to be encoded):

RUNval	*RUNcnt*	*RUNindex*	*RItype*	*Errval* (mod)	*TEMP*	*k*	*map*	*EMErrval*	*Q*	*A*[*Q*]	*N*[*Q*]	*Nn*[*Q*]
145	2	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

And hence the output bits are 1 1

The parameter values after updating are:

RUNindex	*A*[*Q*]	*N*[*Q*]	*Nn*[*Q*]
2	N/A	N/A	N/A

So, the JPEG-LS coded data segment, without any marker segments, should be:

Binary				Hexadecimal
1100 0000	0000 0000	0000 0000	0110 1100	C0 00 00 6C
1000 0000	0010 0000	1000 1110	0000 0001	80 20 8E 01
1100 0000	0000 0000	0000 0000	0101 0111	C0 00 00 57
0100 0000	0000 0000	0000 0000	0110 1110	40 00 00 6E
1110 0110	0000 0000	0000 0000	0000 0001	E6 00 00 01
1011 1100	0001 1000	0000 0000	0000 0000	BC 18 00 00
0000 0101	1101 1000	0000 0000	0000 0000	05 D8 00 00
1001 0001	011*0 0000*			91 60

The last five bits (italicised) in the above table are padding.

H.4 Example image data

H.4.1 Bit stream from Example H.3

The complete compressed image data from Example H.3 is shown in Figure H.2:

```
FF D8 FF F7 00 0B 08 00 04 00 04 01 01 11 00 FF
DA 00 08 01 01 00 00 00 00 C0 00 00 6C 80 20 8E
01 C0 00 00 57 40 00 00 6E E6 00 00 01 BC 18 00
00 05 D8 00 00 91 60 FF D9
```

Figure H.2 – Compressed image data from Example H.3

Figure H.3 is a detailed description of this data.

NOTE – In these examples, the compressed image data is given in hexadecimal form. In the figures explaining this data, the data to the left of the '#' symbol is given in hexadecimal notation, text following the '#' symbol is a comment. Any numbers in a comment are given as decimal values.

```
FF D8    # Start of image (SOI) marker
FF F7    # Start of JPEG-LS frame (SOF55) marker – marker segment follows
00 0B    # Length of marker segment = 11 bytes including the length field
08       # P = Precision = 8 bits per sample
00 04    # Y = Number of lines = 4
00 04    # X = Number of columns = 4
01       # Nf = Number of components in the frame = 1
01       # C1 = Component ID = 1 (first and only component)
11       # Sub-sampling: H1 = 1, V1 = 1
00       # Tq1 = 0 (this field is always 0)

FF DA    # Start of scan (SOS) marker
00 08    # Length of marker segment = 8 bytes including the length field
01       # Ns = Number of components for this scan = 1
01       # Ci = Component ID = 1
00       # Tm1 = Mapping table index = 0 (no mapping table)
00       # NEAR = 0 (lossless)
00       # ILV = 0 (interleave mode = non-interleaved)
00       # Al = 0, Ah = 0 (no point transform)

C0 00 ... 91 60 # 30 bytes of compressed image data

FF D9    # End of image (EOI) marker
```

Figure H.3 – Explanation of compressed image data from Example H.3

In this example, the compressed image data has a length of 36 bytes, and the marker segment bytes have a length of 27 bytes for a total of 63 bytes.

H.4.2 Interleaved data

Figure H.4 is based on the data from file T8SSE3.JLS (Test 8 in Annex E). This is a three-component image, with 256 lines and 256 columns (196 623 bytes of source data). The first component is not sub-sampled, whilst the second component is sub-sampled in lines by a factor of four and the third component is sub-sampled in lines and in columns by a factor of two. The image data consists of a single scan, encoded in line interleaved mode with default JPEG-LS preset coding parameters. Near-lossless coding, **NEAR** = 3, has been used.

```
FF D8 FF F7 00 11 08 01 00 01 00 03 01 24 00 02
21 00 03 12 00 FF DA 00 0C 03 01 00 02 00 03 00
03 01 00 00 01 ..............................
.............................................
........ 6B A8 FF D9
```

Figure H.4 – Compressed image data from conformance Test 8

Figure H.5 is a detailed description of this data.

FF D8 # Start of image (**SOI**) marker
FF F7 # Start of JPEG-LS frame (**SOF_{55}**) marker – marker segment follows
00 11 # Length of marker segment = 17 bytes including the length field
08 # **P** = Precision = 8 bits per sample
01 00 # **Y** = Number of lines = 256
01 00 # **X** = Number of columns = 256
03 # **Nf** = Number of components in the frame = 3
01 # $\mathbf{C_1}$ = Component ID = 1 ("red" component)
24 # Sub-sampling for component 1: $\mathbf{H_1}$ = 2, $\mathbf{V_1}$ = 4
00 # $\mathbf{Tq_1}$ = 0 (this field is always 0)
02 # $\mathbf{C_2}$ = Component ID = 2 ("green" component)
21 # Sub-sampling for component 2: $\mathbf{H_2}$ = 2, $\mathbf{V_2}$ = 1
00 # $\mathbf{Tq_2}$ = 0 (this field is always 0)
03 # $\mathbf{C_3}$ = Component ID = 3 ("blue" component)
12 # Sub-sampling for component 3: $\mathbf{H_3}$ = 1, $\mathbf{V_3}$ = 2
00 # $\mathbf{Tq_3}$ = 0 (this field is always 0)

FF DA # Start of scan (**SOS**) marker
00 0C # Length of marker segment = 12 bytes including the length field
03 # **Ns** = Number of components for this scan = 3
01 # $\mathbf{C_1}$ = Component ID = 1
00 # $\mathbf{Tm_1}$ = Mapping table index = 0 (no mapping table)
02 # $\mathbf{C_2}$ = Component ID = 2
00 # $\mathbf{Tm_2}$ = Mapping table index = 0 (no mapping table)
03 # $\mathbf{C_3}$ = Component ID = 3
00 # $\mathbf{Tm_3}$ = Mapping table index = 0 (no mapping table)
03 # **NEAR** = 3 (near-lossless maximum error)
01 # **ILV** = 1 (interleave mode = line interleave)
00 # **Al** = 0, **Ah** = 0 (no point transform)

00 01 … 6B A8 # 32194 bytes of compressed image data

FF D9 # End of image (**EOI**) marker

Figure H.5 – Explanation of compressed image data from conformance Test 8

In this example, the compressed image data has a length of 32 194 bytes, and the marker segment bytes have a length of 37 bytes for a total of 32 231 bytes.

H.4.3 Non-interleaved data

This example is the same as that in H.4.2, but each component is encoded in a separate scan with no interleaving (see Figure H.6).

```
FF D8 FF F7 00 11 08 01 00 01 00 03 01 24 00 02
21 00 03 12 00 FF DA 00 08 01 01 00 03 00 00 00
01 ..........................................
..............................................
..... 08 80 FF DA 00 08 01 02 00 03 00 00 00 00
..............................................
.................. 62 00 FF DA 00 08 01 03 00 03
00 00 00 01 ..................................
..............................................
79 64 A0 FF D9
```

Figure H.6 – Non-interleaved compressed image data

Figure H.7 is a detailed description of this data.

FF D8 # Start of image (**SOI**) marker
FF F7 # Start of JPEG-LS frame (**SOF_{55}**) marker – marker segment follows
00 11 # Length of marker segment = 17 bytes including the length field
08 # **P** = Precision = 8 bits per sample
01 00 # **Y** = Number of lines = 256
01 00 # **X** = Number of columns = 256
03 # **Nf** = Number of components in the frame = 3
01 # **C_1** = Component ID = 1
24 # Sub-sampling for component 1: **H_1** = 2, **V_1** = 4
00 # **Tq_1** = 0 (this field is always 0)
02 # **C_2** = Component ID = 2
21 # Sub-sampling for component 2: **H_2** = 2, **V_2** = 1
00 # **Tq_2** = 0 (this field is always 0)
03 # **C_3** = Component ID = 3
12 # Sub-sampling for component 3: **H_3** = 1, **V_3** = 2
00 # **Tq_3** = 0 (this field is always 0)

FF DA # Start of scan (**SOS**) marker
00 08 # Length of marker segment = 8 bytes including the length field
01 # **Ns** = Number of components for this scan = 1
01 # **C_1** = Component ID = 1
00 # **Tm_1** = Mapping table index = 0 (no mapping table)
03 # **NEAR** = 3 (near-lossless maximum error)
00 # **ILV** = 0 (interleave mode = non-interleaved)
00 # **Al** = 0, **Ah** = 0 (no point transform)

00 01 ... 08 80 # 206 77 bytes of compressed image data

FF DA # Start of scan (**SOS**) marker
00 08 # Length of marker segment = 8 bytes including the length field
01 # **Ns** = Number of components for this scan = 1
02 # **C_1** = Component ID = 2
00 # **Tm_1** = Mapping table index = 0 (no mapping table)
03 # **NEAR** = 3 (near-lossless maximum error)
00 # **ILV** = 0 (interleave mode = non-interleaved)
00 # **Al** = 0, **Ah** = 0 (no point transform)

00 00 ... 62 00 # 5658 bytes of compressed image data

FF DA # Start of scan (**SOS**) marker
00 08 # Length of marker segment = 8 bytes including the length field
01 # **Ns** = Number of components for this scan = 1
03 # **C_1** = Component ID = 3
00 # **Tm_1** = Mapping table index = 0 (no mapping table)
03 # **NEAR** = 3 (near-lossless maximum error)
00 # **ILV** = 0 (interleave mode = non-interleaved)
00 # **Al** = 0, **Ah** = 0 (no point transform)

00 01 ... 64 A0 # 6257 bytes of compressed image data

FF D9 # End of image (**EOI**) marker

Figure H.7– Explanation of non-interleaved compressed image data

In this example, the compressed image data has a length of 32 592 bytes, and the marker segment bytes have a length of 53 bytes for a total of 32 645 bytes.

H.4.4 Conformance data

Figure H.8 is based on the data from file T8NDE3.JLS (Test 10 in Annex E). This is a single-component image, with 128 lines and 128 columns (16 399 bytes of source data). The image data has been encoded with near-lossless coding, **NEAR** = 3, and uses non-default parameters: **T1** = **T2** = **T3** = 9, **RESET** = 31.

```
FF D8 FF F7 00 0B 08 00 80 00 80 01 01 11 00 FF
F8 00 0D 01 00 FF 00 09 00 09 00 09 00 1F FF DA
00 08 01 01 00 03 00 00 00 01 ..................
........................................................
.............................. 04 80 FF D9
```

Figure H.8 – Compressed image data from conformance Test 10

Figure H.9 is a detailed description of this data.

FF D8 # Start of image (**SOI**) marker
FF F7 # Start of JPEG-LS frame (**SOF_{55}**) marker – marker segment follows
00 0B # Length of marker segment = 11 bytes including the length field
08 # **P** = Precision = 8 bits per sample
00 80 # **Y** = Number of lines = 128
00 80 # **X** = Number of columns = 128
01 # **Nf** = Number of components in the frame = 1
01 # **C_1** = Component ID = 1 (first and only component)
11 # Sub-sampling: **H_1** = 1, **V_1** = 1
00 # **Tq_1** = 0 (this field is always 0)

FF F8 # **LSE** – JPEG-LS preset parameters marker
00 0D # Length of marker segment = 13 bytes including the length field
01 # **ID** = 1, JPEG-LS preset coding parameters
00 FF # **MAXVAL** = 255
00 09 # **T1** = 9
00 09 # **T2** = 9
00 09 # **T3** = 9
00 1F # **RESET** = 31

FF DA # Start of scan (**SOS**) marker
00 08 # Length of marker segment = 8 bytes including the length field
01 # **Ns** = Number of components for this scan = 1
01 # **C_1** = Component ID = 1
00 # **Tm_1** = Mapping table index = 0 (no mapping table)
03 # **NEAR** = 3 (near-lossless maximum error)
00 # **ILV** = 0 (interleave mode = non-interleaved)
00 # **Al** = 0, **Ah** = 0 (no point transform)

00 01 ... 04 80 # 6069 bytes of encoded data

FF D9 # End of image (**EOI**) marker

Figure H.9 – Explanation of compressed image data from conformance Test 10

In this example, the compressed image data has a length of 6069 bytes, and the marker segment bytes have a length of 42 bytes for a total of 6111 bytes.

H.4.5 Example of a palletised image

This example assumes a palletised image referenced as indices (4 lines × 3 columns over a 4-symbol alphabet), indexed as shown in Figure H.10:

```
00 00 01
01 01 02
02 02 03
03 03 03
```

Figure H.10 – Indexed data for palletised image

The sample mapping table is as indicated in Figure H.11:

Index	RGB triplet
00	FF FF FF
01	FF 00 00
02	00 FF 00
03	00 00 FF

Figure H.11 – Sample mapping table for palletised image

Figure H.12 shows the compressed image data.

```
FF D8 FF F7 00 0B 02 00 04 00 03 01 01 11 00 FF
F8 00 11 02 05 03 FF FF FF FF 00 00 00 FF 00 00
00 FF FF DA 00 08 01 01 05 00 00 00 DB 95 F0 FF
D9
```

Figure H.12 – Compressed data for palletised image

Figure H.13 is a detailed description of this data.

```
FF D8     # Start of image (SOI) marker
FF F7     # Start of JPEG-LS frame (SOF55) marker – marker segment follows
00 0B     # Length of marker segment = 11 bytes including the length field
02        # P = Precision = 2 bits per sample
00 04     # Y = Number of lines = 4
00 03     # X = Number of columns = 3
01        # Nf = Number of components in the frame = 1
01        # C1 = Component ID = 1 (first and only component)
11        # Sub-sampling: H1 = 1, V1 = 1
00        # Tq1 = 0 (this field is always 0)

FF F8     # LSE – JPEG-LS preset parameters marker
00 11     # Length of marker segment = 17 bytes including the length field
02        # ID = 2, mapping table
05        # TID = 5 Table identifier (arbitrary)
03        # Wt = 3 Width of table entry
FF FF FF  # Entry for index 0
FF 00 00  # Entry for index 1
00 FF 00  # Entry for index 2
00 00 FF  # Entry for index 3

FF DA     # Start of scan (SOS) marker
00 08     # Length of marker segment = 8 bytes including the length field
01        # Ns = Number of components for this scan = 1
01        # C1 = Component ID = 1
05        # Tm1 = Mapping table identifier = 5
00        # NEAR = 0 (near-lossless max error)
00        # ILV = 0 (interleave mode = non-interleaved)
00        # Al = 0, Ah = 0 (no point transform)

DB 95 F0  # 3 bytes of compressed image data

FF D9     # End of image (EOI) marker
```

Figure H.13 – Explanation of compressed data for palletised image

H.5 Use of SPIFF with JPEG-LS compressed image data

H.5.1 Example of minimum SPIFF file header

An example of the minimum SPIFF (see ITU-T Rec. T.84 | ISO/IEC 10918-3) file header is given below in Table H.1. The use of SPIFF is non-mandatory, but recommended for applications which transfer compressed image data between application environments. Although the example below gives a minimal SPIFF compliant file, implementers should be aware that for conformance with SPIFF, file decoders need to be able to parse the SPIFF directory entries correctly, even if no action is taken with respect to their content.

The example is based on SPIFF as amended in ITU-T Rec. T.84 | ISO/IEC 10918-3 Amd.1. This includes a number of changes to the original version, in particular in introducing a new value of the parameter **C**, compression type specifically for JPEG-LS as defined in this Recommendation | International Standard.

NOTE – For the definition of symbols in Table H.1, see ITU-T T.84 | ISO/IEC 10918-3 Amd.1.

Table H.1 – Example of minimum SPIFF file format

Parameter	Description	Values
MN	Magic number	X'FFD8FFE8'
HLEN	Length of header, 32 bytes	X'0020'
IDENT	Identifies 'SPIFF'	X'535049464600'
VERS	Version number (2.0)	X'0200'
P	Profile ID (not specified)	X'00'
NC	Number of components (3)	X'03'
HEIGHT	Height (2048)	X'00000800'
WIDTH	Width (3072)	X'00000C00'
S	Colour space (uncalibrated RGB)	X'0A'
BPS	Bits per sample (8)	X'08'
C	Compression type (JPEG-LS)	X'06'
R	Resolution units (dpi)	X'01'
VRES	Vertical resolution (300 dpi)	X'0000012C'
HRES	Horizontal resolution (300 dpi)	X'0000012C'
variable	Zero or more directory entries, comprising a header (X'FFE8'), a length count (2 bytes), an identifier (4 bytes), and contents (variable)	*varies, can be omitted*
EMN	EMN for End of Directory tag	X'FFE8'
EODLEN	EOD length, including SOI (8)	X'0008'
EODTAG	EOD tag identifier (1)	X'00000001'
SOI	Start of image	X'FFD8'
variable	*Image data*	*varies*
EOI	End of image	X'FFD9'

Annex I

Bibliography

(This annex does not form an integral part of this Recommendation | International Standard)

GALLAGER and VOORHIS (D.V.), Optimal source codes for geometrically distributed integer alphabets, *IEEE Transactions on Information Theory,* Vol. 21, pp. 228-230, March 1975.

GOLOMB (S.W.), Run-length encodings, *IEEE Transactions on Information Theory,* Vol. 12, pp. 399-401, July 1966.

HUFFMAN (D.A.), A method for the construction of minimum redundancy codes, *Proceedings IRE,* Vol. 40, pp. 1098-1101, 1952.

LANGDON, Jr. (G.), An adaptive run-length coding algorithm, *IBM Technical Disclosure Bulletin,* Vol. 26, pp. 3783-3785, December 1983.

LANGDON (G.G.), Sunset: A hardware-oriented algorithm for lossless compression of gray-scale images, *Proceedings SPIE Medical Imaging V: Image Capture, Formatting, Display*, Vol. 1444, pp. 272-282, May 1991.

MARTUCCI (S.A.), Reversible compression of HDTV images using median adaptive prediction and arithmetic coding, *Proceedings IEEE International Symposium on Circuits and Systems,* pp. 1310-1313, 1990.

MEMON (N.D.) and SAYOOD (K.), Lossless image compression: A comparative study, *Proceedings SPIE,* Vol. 2418, pp. 8-20, February 1995.

MERHAV (N.), SEROUSSI (G.) and WEINBERGER (M.J.), Modeling and low-complexity adaptive coding for image prediction residuals, *Proceedings IEEE International Conference on Image Processing 96*, Vol. II, pp. 353-356, Lausanne, Switzerland, September 1996.

NETRAVALI (A.) and LIMB (J.O.), Picture coding: A Review, *Proceedings of IEEE,* Vol. 68, pp. 366-406, 1980.

ONO (F.), KINO (S.), YOSHIDA (M.) and KIMURA (T.), Bi-level image coding with Melcode – Comparison of block type code and arithmetic type code, *Proceedings of Globecom 89*, pp. 255-60, November 1989.

PENNEBAKER (W.B.) and MITCHELL (J.L.), *JPEG: Still Image Data Compression Standard,* Van Nostrand Reinhold, New York, 1993.

RABBANI (M.) and JONES (P.), *Digital Image Compression Techniques,* Tutorial Texts in Optical Engineering, Vol. TT7, SPIE Press, 1991.

RICE (R.F.), Some practical universal noiseless coding techniques, *Tech. Report JPL,* Vol. 79-22, Jet Propulsion Laboratory, Pasadena, CA, March 1979.

RICE (R.F.), Some practical universal noiseless coding techniques: III, *Tech. Report JPL*, Vol. 91-3, Jet Propulsion Laboratory, Pasadena, CA, November 1991.

RISSANEN (J.), A universal data compression system, *IEEE Transactions on Information Theory,* Vol. 29, pp. 656-664, 1983.

RISSANEN (J.), Universal coding, information, prediction, and estimation, *IEEE Transactions on Information Theory*, Vol. 30, pp. 629-636, 1984.

RISSANEN (J.) and LANGDON, Jr. (G.G.); Universal modelling and coding, *IEEE Transactions on Information Theory*, Vol. 27, pp. 12-23, 1981.

TEUHOLA (J.), A compression method for clustered bit-vectors, *Information Proc. Letters*, Vol. 7, pp. 308-311, 1978.

TODD (S.), LANGDON (G.G.) and RISSANEN (J.), Parameter reduction and context selection for compression of gray-scale images, *IBM Journal of Research and Development*, Vol. 29 (2), pp. 188-193, 1985.

WEINBERGER (M.J.), RISSANEN (J.) and ARPS (R.), Applications of universal context modelling to lossless compression of gray-scale images, *IEEE Transactions on Image Processing,* Vol. 5, pp. 575-586, April 1996.

WEINBERGER (M.J.), SEROUSSI (G.) and SAPIRO (G.), LOCO-I: A low complexity, context-based, lossless image compression algorithm, *Proceedings Data Compression Conference,* Snowbird, Utah, pp. 140-149, April 1996.

WU (X.), An algorithmic study in lossless image compression, *Proceedings of the 1996 Data Compression Conference,* Snowbird, Utah, pp. 150-159, April 1996.

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	**Terminals for telematic services**
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure
Series Z	Languages and general software aspects for telecommunication systems