
Information technology — Lossy/ lossless coding of bi-level images

*Technologies de l'information — Codage avec ou sans perte des
images au trait*



Reference number
ISO/IEC 14492:2019(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2019



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

FOREWORD

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see <http://patents.iec.ch>).

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

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

This document was prepared by ITU-T as ITU-T Rec. T.88 (2018) and drafted in accordance with its editorial rules. It was adopted by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This second edition cancels and replaces the first edition (ISO/IEC 14492:2001), which has been technically revised. It also incorporates ISO/IEC 14492:2001/Amd.1:2004, ISO/IEC 14492:2001/Amd.2:2003 and ISO/IEC 14492:2001/Amd.3:2012.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Information technology – Lossy/lossless coding of bi-level images

Summary

Rec. ITU-T T.88 | ISO/IEC 14492, commonly known as JBIG2, defines a coding method for bi-level images (e.g., black and white printed matter). These are images consisting of a single rectangular bit plane, with each pixel taking on one of just two possible colours. This Recommendation | International Standard has been explicitly prepared for a lossy, lossless, and lossy-to-lossless image compression.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T T.88	2000-02-10	8	11.1002/1000/4845
1.1	ITU-T T.88 (2000) Amd. 1	2003-06-29	16	11.1002/1000/6388
1.2	ITU-T T.88 (2000) Amd. 2	2003-06-29	16	11.1002/1000/6389
1.3	ITU-T T.88 (2000) Amd. 3	2011-05-14	16	11.1002/1000/11312
2.0	ITU-T T.88	2018-08-29	16	11.1002/1000/13688

Keywords

Bi-level images; image compression; lossy image compression; lossless image compression; lossy-to-lossless image compression; JBIG.

* To access the Recommendation, type the URL <http://handle.itu.int/> in the address field of your web browser, followed by the Recommendation's unique ID. For example, <http://handle.itu.int/11.1002/1000/11830-en>.

FOREWORD

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

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

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

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

NOTE

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

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

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

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2019

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Table of Contents

	Page
0 Introduction	v
0.1 Interpretation and use of the requirements	v
0.2 Lossy coding	ix
1 Scope	1
2 Normative references	1
3 Definitions	1
4 Symbols and abbreviations	3
4.1 Abbreviations	3
4.2 Symbol definitions	4
4.3 Operator definitions	11
5 Conventions	11
5.1 Typographic conventions	11
5.2 Binary notation	11
5.3 Hexadecimal notation	11
5.4 Integer value syntax	11
5.5 Array notation and conventions	12
5.6 Image and bitmap conventions	12
6 Decoding Procedures	13
6.1 Introduction to decoding procedures	13
6.2 Generic region decoding procedure	14
6.3 Generic Refinement Region Decoding Procedure	21
6.4 Text Region Decoding Procedure	25
6.5 Symbol Dictionary Decoding Procedure	33
6.6 Halftone Region Decoding Procedure	41
6.7 Pattern Dictionary Decoding Procedure	44
6.8 Colour palette decoding procedure	46
7 Control Decoding Procedure	47
7.1 General description	47
7.2 Segment header syntax	48
7.3 Segment types	52
7.4 Segment syntaxes	54
8 Page Make-up	82
8.1 Decoder model	82
8.2 Page image composition	83
9 Encoding procedures (informative)	85
10 Control encoding procedures (informative)	85
11 Page break-up (informative)	85
11.1 Page break-up architecture	86
11.2 Page image decomposition	86
11.3 Multi-page document composition	88
Annex A – Arithmetic integer decoding procedure	89
A.1 General description	89
A.2 Procedure for decoding values (except IAID)	89
A.3 The IAID decoding procedure	91
Annex B – Huffman table decoding procedure	93
B.1 General description	93
B.2 Code table structure	93
B.3 Assigning the prefix codes	94
B.4 Using a Huffman table	95
B.5 Standard Huffman tables	96

	Page
Annex C – Gray-scale image decoding procedure.....	103
C.1 General description	103
C.2 Input parameters.....	103
C.3 Return value	103
C.4 Variables used in decoding	103
C.5 Decoding the gray-scale image	103
Annex D – File formats	105
D.1 Sequential organization.....	105
D.2 Random-access organization.....	105
D.3 Embedded organization	106
D.4 File header syntax	106
Annex E – Arithmetic coding	108
E.1 Binary encoding	108
E.2 Description of the arithmetic encoder	109
E.3 Arithmetic decoding procedure.....	116
Annex F – Profiles	124
Annex G – Arithmetic decoding procedure (software conventions).....	127
Annex H – Datastream example and test sequence	129
H.1 Datastream example.....	129
H.2 Test sequence for arithmetic coder	150
Annex I – Patents.....	156
I.1 List of patents.....	156
I.2 Contact addresses for patent information.....	157
Annex J – Compliant example encoding methods.....	158
J.1 List of JBIG2 encoding components and corresponding algorithms	158
J.2 Method references.....	159
Annex K – Electronic conformance data and sample software	161
K.1 Attached electronic data (informative).....	161
K.2 Working environments of the released sample software (informative)	162
K.3 How to use the sample software (informative)	162
Bibliography	165

Electronic attachments:

1. Reference source code implementation
2. Test bitstreams

0 Introduction

This Recommendation | International Standard, informally called JBIG2, defines a coding method for bi-level images (e.g., black and white printed matter). These are images consisting of a single rectangular bit plane, with each pixel taking on one of just two possible colours. Multiple colours are to be handled using an appropriate higher level standard such as Recommendation ITU-T T.44. It is being drafted by the Joint Bi-level Image Experts Group (JBIG), a "Collaborative Team", established in 1988, that reports both to ISO/IEC JTC 1/SC29/WG1 and to ITU-T.

Compression of this type of image is also addressed by existing facsimile standards, for example by the compression algorithms in Recommendations ITU-T T.4 (MH, MR), T.6 (MMR), T.82 (JBIG1), and T.85 (Application profile of JBIG1 for facsimile). Besides the obvious facsimile application, JBIG2 will be useful for document storage and archiving, coding images on the World Wide Web, wireless data transmission, print spooling, and even teleconferencing.

As the result of a process that ended in 1993, JBIG produced a first coding standard formally designated Recommendation ITU-T T.82 | International Standard ISO/IEC 11544, which is informally known as JBIG or JBIG1. JBIG1 is intended to behave as lossless and progressive (lossy-to-lossless) coding. Though it has the capability of lossy coding, the lossy images produced by JBIG1 have significantly lower quality than the original images because the number of pixels in the lossy image cannot exceed one quarter of those in the original image.

On the contrary, JBIG2 was explicitly prepared for lossy, lossless, and lossy-to-lossless image compression. The design goal for JBIG2 was to allow for lossless compression performance better than that of the existing standards, and to allow for lossy compression at much higher compression ratios than the lossless ratios of the existing standards, with almost no visible degradation of quality. In addition, JBIG2 allows both quality-progressive coding, with the progression going from lower to higher (or lossless) quality, and content-progressive coding, successively adding different types of image data (for example, first text, then halftones). A typical JBIG2 encoder decomposes the input bi-level image into several regions and codes each of the regions separately using a different coding method. Such content-based decomposition is very desirable especially in interactive multimedia applications. JBIG2 can also handle a set of images (multiple page document) in an explicit manner.

As is typical with image compression standards, JBIG2 explicitly defines the requirements of a compliant bitstream, and thus defines decoder behaviour. JBIG2 does not explicitly define a standard encoder, but instead is flexible enough to allow sophisticated encoder design. In fact, encoder design will be a major differentiator among competing JBIG2 implementations.

Although this Recommendation | International Standard is phrased in terms of actions to be taken by decoders to interpret a bitstream, any decoder that produces the correct result (as defined by those actions) is compliant, regardless of the actions it actually takes.

Annexes A, B, C, D, E and F are normative, and thus form an integral part of this Recommendation | International Standard. Annexes G, H, I, J and K are informative, and thus do not form an integral part of this Recommendation | International Standard.

0.1 Interpretation and use of the requirements

This section is informative and designed to aid in interpreting the requirements of this Recommendation | International Standard. The requirements are written to be as general as possible to allow a large amount of implementation flexibility. Hence the language of the requirements is not specific about applications or implementations. In this section a correspondence is drawn between the general wording of the requirements and the intended use of this Recommendation | International Standard in typical applications.

0.1.1 Subject matter for JBIG2 coding

JBIG2 is used to code bi-level documents. A bi-level document contains one or more pages. A typical page contains some text data, that is, some characters of a small size arranged in horizontal or vertical rows. The characters in the text part of a page are called *symbols* in JBIG2. A page may also contain "halftone data", that is, gray-scale or colour multi-level images (e.g., photographs) that have been dithered to produce bi-level images. The periodic bitmap cells in the halftone part of the page are called *patterns* in JBIG2. In addition, a page may contain other data, such as line art and noise. Such non-text, non-halftone data is called *generic* data in JBIG2.

The JBIG2 image model treats text data and halftone data as special cases. It is expected that a JBIG2 encoder will divide the content of a page into a text region containing digitized text, a halftone region containing digitized halftones, and a generic region containing the remaining digitized image data, such as line-art. In some circumstances, it is better (in image quality or compressed data size) to consider text or halftones as generic data; conversely, in some circumstances it is better to consider generic data using one of the special cases.

An encoder is permitted to divide a single page into any number of regions, but often three regions will be sufficient, one for textual symbols, one for halftone patterns, and the third for the generic remainder. In some cases, not all types of data may be present, and the page may consist of fewer than three regions.

The various regions may overlap on the physical page. JBIG2 provides the means to specify how the overlapping regions are recombined to form the final page image.

A text region consists of a number of symbols placed at specified locations on a background. The symbols usually correspond to individual text characters. JBIG2 obtains much of its effectiveness by using individual symbols more than once. To reuse a symbol, an encoder or decoder must have a succinct way of referring to it. In JBIG2, the symbols are collected into one or more symbol dictionaries. A symbol dictionary is a set of bitmaps of text symbols, indexed so that a symbol bitmap may be referred to by an index number.

A halftone region consists of a number of patterns placed along a regular grid. The patterns usually correspond to gray-scale values. Indeed, the coding method of the pattern indices is designed as a gray-scale coder. Compression can be realized by representing the binary pixels of one grid cell by a single integer, the halftone index (which is usually a rendered gray-scale value). This many-to-one mapping (the pattern in a cell into a gray-scale value) may have the effect that edge information present in the original bitmap is lost by halftone coding. For this reason, lossless or near-lossless coding of halftones will often be better in image quality (though larger in size) if the halftone is coded with generic coding rather than halftone coding.

0.1.2 Relationship between segments and documents

A JBIG2 file contains the information needed to decode a bi-level document. A JBIG2 file is composed of *segments*. A typical page is coded using several segments. In a simple case, there will be a page information segment, a symbol dictionary segment, a text region segment, a pattern dictionary segment, a halftone region segment, and an end-of-page segment. The page information segment provides general information about the page, such as its size and resolution. The dictionary segments collect bitmaps referred to in the region segments. The region segments describe the appearance of the text and halftone regions by referencing bitmaps from a dictionary and specifying where they should appear on the page. The end-of-page segment marks the end of the page.

0.1.3 Structure and use of segments

Each segment contains a segment header, a data header, and data. The segment header is used to convey segment reference information and, in the case of multi-page documents, page association information. A data header gives information used for decoding the data in the segment. The data describes an image region or a dictionary, or provides other information.

Segments are numbered sequentially. A segment may refer to a lower-numbered, or *earlier*, segment. A region segment is always associated with one specific page of the document. A dictionary segment may be associated with one page of the document, or it may be associated with the document as a whole.

A region segment may refer to one or more earlier dictionary segments. The purpose of such a reference is to allow the decoder to identify symbols in a dictionary segment that are present into the image.

A region segment may refer to an earlier region segment. The purpose of such a reference is to combine the image described by the earlier segment with the current representation of the page.

A dictionary segment may refer to earlier dictionary segments. The symbols added to a dictionary segment may be described directly, or may be described as refinements of symbols described previously, either in the same dictionary segment or in earlier dictionary segments.

A JBIG2 file may be organized in two ways, sequential or random access. In the sequential organization, each segment's segment header immediately precedes that segment's data header and data, all in sequential order. In the random access organization, all the segment headers are collected together at the beginning of the file, followed by the data (including data headers) for all the segments, in the same order. This second organization permits a decoder to determine all segment dependencies without reading the entire file.

A third way of encapsulating of JBIG2-encoded data is to embed it in a non-JBIG2 file – this is sometimes called the *embedded organization*. In this case a different file format carries JBIG2 segments. The segment header, data header, and data of each segment are stored together, but the embedding file format may store the segments in any order, at any set of locations within its own structure.

0.1.4 Internal representations

Decoded data must be stored before printing or display. While this Recommendation | International Standard does not specify how to store it, its decoding model presumes certain data structures, specifically buffers and dictionaries.

Figure 1 illustrates major decoder components and associated buffers. In this figure, decoding procedures are outlined in bold lines, and memory components are outlined in non-bold lines. Also, bold arrows indicate that one decoding procedure invokes another decoding procedure; for example, the symbol dictionary decoding procedure invokes the generic region decoding procedure to decode the bitmaps for the symbols that it defines. Non-bold arrows indicate flow of data: the text region decoding procedure reads symbols from the symbol memory and draws them into the page buffer or an auxiliary buffer. Although it is not shown in Figure 1, the encoded data stream flows to the decoding procedures, and the block labelled "Page and auxiliary buffers" produces the final decoded page images.

Figure 1 – Block diagram of major decoder components

The resources required to decode any given JBIG2 bitstream depend on the complexity of that bitstream. Some techniques such as striping can be used to reduce decoder memory requirements. It is estimated that a full-featured decoder may need two full-page buffers, plus about the same amount of dictionary memory, plus about 100 kilobytes of arithmetic coding context memory, to decode most bitstreams.

A buffer is a representation of a bitmap. A buffer is intended to hold a large amount of data, typically the size of a page. A buffer may contain the description of a region or of an entire page. Even if the buffer describes only a region, it has information associated with it that specifies its placement on the page. Decoding a region segment modifies the contents of a buffer.

There is one special buffer, the *page buffer*. It is intended that the decoder accumulate region data directly in the page buffer until the page has been completely decoded; then the data can be sent to an output device or file. Decoding an *immediate* region segment modifies the contents of the page buffer. The usual way of preparing a page is to decode one or more immediate region segments, each one modifying the page buffer. The decoder may output an incomplete page buffer, either as part of progressive transmission or in response to user input. Such output is optional, and its content is not specified by this Recommendation | International Standard.

All other buffers are auxiliary buffers. It is intended that the decoder fill an auxiliary buffer, then later use it to refine the page buffer. In an application, it will often be unnecessary to have any auxiliary buffers. Decoding an *intermediate* region segment modifies the contents of an auxiliary buffer. The decoder may use auxiliary buffers to output pages other than those found in a complete page buffer, either as part of progressive transmission or in response to user input. Such output is optional, and its content is not specified by this Recommendation | International Standard.

A symbol dictionary consists of an indexed set of bitmaps. The bitmaps in a dictionary are typically small, approximately the size of text characters. Unlike a buffer, a bitmap in a dictionary does not have page location information associated with it.

0.1.5 Decoding results

Decoding a segment involves invocation of one or more decoding procedures. The decoding procedures to be invoked are determined by the segment type.

The result of decoding a region segment is a bitmap stored in a buffer, possibly the page buffer. Decoding a region segment may fill a new buffer, or may modify an existing buffer. In typical applications, placing the data into a buffer involves changing pixels from the background colour to the foreground colour, but this Recommendation | International Standard specifies other permissible ways of changing a buffer's pixels.

A typical page will be described by one or more immediate region segments, each one resulting in modification of the page buffer.

Just as it is possible to specify a new symbol in a dictionary by refining a previously specified symbol, it is also possible to specify a new buffer by refining an existing buffer. However, a region may be refined only by the generic refinement decoding procedure. Such a refinement does not make use of the internal structure of the region in the buffer being refined. After a buffer has been refined, the original buffer is no longer available.

The result of decoding a dictionary segment is a new dictionary. The symbols in the dictionary may later be placed into a buffer by the text region decoding procedure.

0.1.6 Decoding procedures

The *generic region decoding procedure* fills or modifies a buffer directly, pixel-by-pixel if arithmetic coding is being used, or by runs of foreground and background pixels if MMR and Huffman coding are being used. In the arithmetic coding case, the prediction context contains only pixels determined by data already decoded within the current segment.

The *generic refinement region decoding procedure* modifies a buffer pixel-by-pixel using arithmetic coding. The prediction context uses pixels determined by data already decoded within the current segment as well as pixels already present either in the page buffer or in an auxiliary buffer.

The *text region decoding procedure* takes symbols from one or more symbol dictionaries and places them in a buffer. This procedure is invoked during the decoding of a text region segment. The text region segment contains the position and index information for each symbol to be placed in the buffer; the bitmaps of the symbols are taken from the symbol dictionaries.

The *symbol dictionary decoding procedure* creates a symbol dictionary, that is, an indexed set of symbol bitmaps. A bitmap in the dictionary may be coded directly; it may be coded as a refinement of a symbol already in a dictionary; or it may be coded as an aggregation of two or more symbols already in dictionaries. This decoding procedure is invoked during the decoding of a symbol dictionary segment.

The *halftone region decoding procedure* takes patterns from a pattern dictionary and places them in a buffer. This procedure is invoked during the decoding of a halftone region segment. The halftone region segment contains the position information for all the patterns to be placed in the buffer, as well as index information for the patterns themselves. The patterns, the fixed-size bitmaps of the halftone, are taken from the halftone dictionaries.

The *pattern dictionary decoding procedure* creates a dictionary, that is, an indexed set of fixed-size bitmaps (patterns). The bitmaps in the dictionary are coded directly and jointly. This decoding procedure is invoked during the decoding of a pattern dictionary segment.

The *control decoding procedure* decodes segment headers, which include segment type information. The segment type determines which decoding procedure must be invoked to decode the segment. The segment type also determines where the decoded output from the segment will be placed. The segment reference information, also present in the segment header and decoded by the control decoding procedure, determines which other segments must be used to decode the current segment. The control decoding procedure affects everything shown in Figure 1, and so is not shown there as a separate block.

Table 1 summarizes the types of data being decoded, which decoding procedure is responsible for decoding them, and what the final representations of the decoded data are.

Table 1 – Entities in the decoding process

Concept	JBIG2 bitstream entity	JBIG2 decoding entity	Physical representation
Document	JBIG2 file	JBIG2 decoder	Output medium or device
Page	Collection of segments	Implicit in control decoding procedure	Page buffer
Region	Region segment	Region decoding procedure	Page buffer or auxiliary buffer
Dictionary	Dictionary segment	Dictionary decoding procedure	List of symbols
Character	Field within a symbol dictionary segment	Symbol dictionary decoding procedure	Symbol bitmap
Gray-scale value	Field within a halftone dictionary segment	Pattern dictionary decoding procedure	Pattern

0.2 Lossy coding

This Recommendation | International Standard does not define how to control lossy coding of bi-level images. Rather it defines how to perform perfect reconstruction of a bitmap that the encoder has chosen to encode. If the encoder chooses to encode a bitmap that is different than the original, the entire process becomes one of lossy coding. The different coding methods allow for different methods of introducing loss in a profitable way.

0.2.1 Symbol coding

Lossy symbol coding provides a natural way of doing lossy coding of text regions. The idea is to allow small differences between the original symbol bitmap and the one indexed in the symbol dictionary. Compression gain is effected by not having to code a large dictionary and, afterwards, by having a cheap symbol index coding as a consequence of the smaller dictionary. It is up to the encoder to decide when two bitmaps are essentially the same or essentially different. This technique was first described in [1].

The hazard of lossy symbol coding is to have *substitution errors*, that is, to have the encoder replace a bitmap corresponding to one character by a bitmap depicting a different character, so that a human reader misreads the character. The risk of substitution errors can be reduced by using intricate measures of difference between bitmaps and/or by making sure that the critical pixels of the indexed bitmap are correct. One way to control this, described in [5], is to index the possibly wrong symbol and then to apply refinement coding to that symbol bitmap. The idea is to encode the basic character shape at little cost, then correct pixels that the encoder believes alter the meaning of the character.

The process of beneficially introducing loss in textual regions may also take simpler forms such as removing flyspecks from documents or regularizing edges of letters. Most likely such changes will lower the code length of the region without affecting the general appearance of the region – possibly even improving the appearance.

A number of examples of performing this sort of lossy symbol coding with JBIG2 can be found in [7].

NOTE – Although the term "text region" is used for regions of the page coded using symbol coding, other possible uses of symbol coding include coding line-art and other non-textual data.

0.2.2 Generic coding

To effect near-lossless coding using generic coding, the encoder applies a preprocess to an original image and encodes the changed image losslessly. The difficulties are to ensure that the changes result in a lower code length and that the quality of the changed image does not suffer badly from the changes. Two possible preprocesses are given in [11]. These preprocesses flip pixels that, when flipped, significantly lower the total code length of the region, but can be flipped without seriously impairing the visual quality. The preprocesses provide for effective near-lossless coding of periodic halftones and for a moderate gain in compression for other data types. The preprocesses are not well-suited for error diffused images and images dithered with blue noise as perceptually lossless compression will not be achieved at a significantly lower rate than the lossless rate.

0.2.3 Halftone coding

Halftone coding is the natural way to obtain very high compression for *periodic* halftones, such as clustered-dot ordered dithered images. In contrast to lossy generic coding as described above, halftone coding does not intend to preserve the original bitmap, although this is possible in special cases. Loss can also be introduced for additional compression by not putting all the patterns of the original image into the dictionary, thereby reducing both the number of halftone patterns and the number of bits required to specify which pattern is used in which location.

For lossy coding of error diffused images and images dithered with blue noise, it is advisable to use halftone coding with a small grid size. A reconstructed image will lack fine details and may display blockiness but will be clearly recognizable. The blockiness may be reduced on the decoder side in a postprocess; for instance, by using other reconstruction patterns than those that appear in the dictionary. Error diffused images may also be coded losslessly, or with controlled loss as described above, using generic coding.

More details on performing this halftone coding can be found in [12].

0.2.4 Consequences of inadequate segmentation

In order to obtain optimum coding, both in terms of quality and file size, the correct form of encoding should be used for the appropriate regions of the document pages. This subclause briefly describes the consequences of errors in this segmentation.

Using lossy symbol coding for a document containing both text and halftone data will result in poor compression. Depending on the encoder, the quality of the halftone data may be good or bad. Using the form of lossy symbol coding described in [5], the visual quality will probably not suffer.

Using lossy generic coding (using the preprocesses given in [11]) for a document containing both symbol and halftone data usually results in good quality and moderate compression.

Line art and regions of handwritten text may be coded efficiently using generic coding, but depending on the encoder, these types of regions can also be very effectively coded with symbol coding.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Lossy/lossless coding of bi-level images

1 Scope

This Recommendation | International Standard¹ defines methods for coding bi-level images and sets of images (documents consisting of multiple pages). It is particularly suitable for bi-level images consisting of text and dithered (halftone) data.

The methods defined permit lossless (bit-preserving) coding, lossy coding, and progressive coding. In progressive coding, the first image is lossy; subsequent images may be lossy or lossless.

This Recommendation | International Standard also defines file formats to enclose the coded bi-level image data.

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 ITU maintains a list of the currently valid ITU-T Recommendations.

- Recommendation ITU-T T.6 (1988), *Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus*.
- Recommendation ITU-T T.45 (2000), *Run-length colour encoding*.
- ISO/IEC 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*.
- ISO/IEC 10646:2017, *Information technology – Universal Coded Character Set (UCS)*.

3 Definitions

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

- 3.1 adaptive template pixel(s):** Special pixel(s), in a template, whose location is not fixed.
- 3.2 aggregation:** Joining or merging of several individual symbols into a new symbol.
- 3.3 bi-level image:** Rectangular array of bits.
- 3.4 bit:** Binary digit, representing the value 0 or 1.
- 3.5 bitmap:** Bi-level image.
- 3.6 buffer:** Storage area used to hold a bitmap.
- 3.7 byte:** Eight bits of data.
- 3.8 combination operator:** Operator used to combine the prior contents of a bitmap with new values being drawn into that bitmap.
- 3.9 coordinate system:** Numbering system for two-dimensional locations where locations are labelled by two numbers, the first one increasing from left to right and the second one increasing from top to bottom.
- 3.10 delta S:** Difference in the S coordinates between two successive symbol instances in a non-empty strip.
- 3.11 delta T:** Difference in the T coordinates between two successive non-empty strips.

¹ This Recommendation | International Standard includes an electronic attachment with a reference source code implementation and test bitstreams.

- 3.12 decoding procedure:** Component of a decoder that decodes a certain type of data.
- 3.12.1 integer decoding procedure:** Decoding procedure whose output on each invocation is a single value.
- 3.12.2 arithmetic integer decoding procedure:** Integer decoding procedure that uses arithmetic entropy decoding.
- 3.12.3 region decoding procedure:** Decoding procedure whose output is a bitmap.
- 3.12.4 generic region decoding procedure:** Region decoding procedure that operates by decoding pixels individually or in runs.
- 3.12.5 generic refinement region decoding procedure:** Region decoding procedure that operates by modifying a reference bitmap to produce an output bitmap.
- 3.12.6 gray-scale decoding procedure:** Decoding procedure whose output is a gray-scale image.
- 3.12.7 pattern dictionary decoding procedure:** Decoding procedure whose output is a list of patterns.
- 3.12.8 halftone region decoding procedure:** Region decoding procedure that operates by drawing a set of patterns into a bitmap, placing the patterns according to a halftone grid.
- 3.12.9 Huffman table decoding procedure:** Decoding procedure whose output is a Huffman table.
- 3.12.10 text region decoding procedure:** Region decoding procedure that operates by drawing a set of symbol instances into a bitmap.
- 3.12.11 symbol dictionary decoding procedure:** Decoding procedure whose output is a list of symbols.
- 3.13 decoder:** Entity capable of decoding a bitstream in conformance with this Recommendation | International Standard.
- 3.14 dictionary:** List of bitmaps.
- 3.14.1 pattern dictionary:** List of patterns.
- 3.14.2 symbol dictionary:** List of symbols.
- 3.15 export flag:** Bit indicating that a symbol is on the export list of a symbol dictionary.
- 3.16 export list:** List of the symbols in a symbol dictionary that may be used by referring to that symbol dictionary.
- 3.17 gray-scale image:** Rectangular array of non-negative integer indices.
- 3.18 gray-scale pixel:** Integer-valued element in a gray-scale image.
- 3.19 halftone grid:** Rectilinear grid of locations specifying where patterns are to be drawn.
- 3.20 height class:** Set of symbols in a symbol dictionary whose heights are all equal.
- 3.21 height class delta height:** Difference in height between two height classes.
- 3.22 height class delta width:** Difference in width between two symbols in a height class.
- 3.23 Huffman table:** Collection of table lines specifying how values are encoded.
- 3.24 lossless coding:** Method of encoding data so that the decoded data are identical to the original data.
- 3.25 lossy coding:** Method of encoding data so that the decoded data differ, ideally only in insignificant ways, from the original data.
- 3.26 ordinal:** Value used as a counter.
- 3.27 out-of-band value:** Non-numeric value that may be produced in place of an integer.
- 3.28 pattern:** Bitmap produced by a pattern dictionary decoding procedure.
- 3.29 pixel:** Element with 0 or 1 as its value in a bitmap.
- 3.30 prefix length:** Length of the Huffman code prefix in a table line.
- 3.31 range length:** Number of additional code bits in a table line.
- 3.32 reference bitmap:** Bitmap used as the reference plane during the refinement region decoding procedure.
- 3.33 referred-to segment:** Other segment required in order to decode the current segment.

- 3.34 region:** Bitmap produced by a region decoding procedure.
- 3.35 segment:** Segment header and its segment data.
- 3.36 strip:** Full-width or full-height portion of the coordinate system of a text region.
- 3.36.1 empty strip:** Strip containing the reference corners of no symbol instances.
- 3.36.2 non-empty strip:** Strip containing the reference corner of at least one symbol instance.
- 3.37 strip size:** Extent in pixels of the non-full dimension of a strip.
- 3.38 symbol:** Bitmap produced by a symbol dictionary decoding procedure.
- 3.39 symbol ID:** Integer used to identify a symbol or to index into an array of symbols to retrieve the symbol.
- 3.40 symbol instance:** Symbol drawn, possibly with refinement, at a particular location in a text region.
- 3.41 symbol instance refinement delta height:** Difference in height between a symbol instance's reference bitmap and the bitmap produced by the generic refinement region decoding procedure.
- 3.42 symbol instance refinement delta width:** Difference in width between a symbol instance's reference bitmap and the bitmap produced by the generic refinement region decoding procedure.
- 3.43 symbol instance refinement delta X:** Difference between the X coordinates of the top left corners of a symbol instance's reference bitmap and the bitmap produced by the generic refinement region decoding procedure.
- 3.44 symbol instance refinement delta Y:** Difference between the Y coordinates of the top left corners of a symbol instance's reference bitmap and the bitmap produced by the generic refinement region decoding procedure.
- 3.45 table line:** Specification of the encoding of a single value or a range of values as a Huffman code prefix followed by a fixed number of additional code bits.
- 3.46 typical prediction:** Method of signalling that an entire row of a generic region is identical to the preceding row.
- 3.47 value:** Integer or out-of-band indicator that is decoded.

4 Symbols and abbreviations

NOTE – Due to ISO nomenclature requirements, within the context of clause 4, the term "symbol" is locally used to mean a variable name.

4.1 Abbreviations

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

AT	Adaptive Template
EOFB	End-of-Facsimile Block
ID	Identifier
LPS	Less Probable Symbol, i.e., less probable binary value
LSB	Least Significant Bit
MMR	Modified Modified READ
MPS	More Probable Symbol, i.e., more probable binary value
MSB	Most Significant Bit
OOB	Out-of-Band
READ	Relative Element Address Designate
TPGD	Typical prediction for generic direct bitmap coding
TPGR	Typical prediction for generic refinement bitmap coding

NOTE – The term "symbol" in the abbreviations LPS and MPS does not refer to the symbols (bitmaps) in this Recommendation | International Standard. The LPS and MPS abbreviations are used despite this because they are the generally-accepted terminology in arithmetic coding.

4.2 Symbol definitions

The symbols used in this Recommendation | International Standard are listed below. A convention is used that parameters to any of the decoding procedures that are used in this Recommendation | International Standard are indicated in **bold face**.

A	Probability interval
a	A real number
ARR	An array
A₁, A₂, A₃, A₄, A₅, A₆, A₇, A₈, A₉, A₁₀, A₁₁, A₁₂	Adaptive template pixels in the generic region decoding procedure
B	Current byte of arithmetically-coded data
B₁	Byte of arithmetically-coded data following the current byte
B_{HC}	A height class collective bitmap in a symbol dictionary decoding procedure
B_{HDC}	A dictionary collective bitmap in a pattern dictionary decoding procedure
B_P	A pattern bitmap in a pattern dictionary decoding procedure
B_S	A symbol bitmap in a symbol dictionary decoding procedure
BM	A bitmap
BP	Pointer to byte B
BPST	Initial value of BP
C	Value of bitstream in code register
Chigh	High-order 16 bits of C
Clow	Low-order 16 bits of C
COLEXTFLAG	A parameter indicating whether the generic region segment is extended to represent a coloured bitmap
CONTEXT	The values of the pixels in a template used in the generic or generic refinement decoding procedure
CPCOMPLEN	The length (in bytes) of each component's value
CPDEFCOLS	The default colour set
CPEXCOLS	The colours defined in the colour palette segment
CPNCOMP	The number of colour components
CPNVALS	The number of colour values coded in this segment
CT	Renormalization shift counter
CURCODE	The Huffman code for the current table line in a Huffman table
CUREXFLAG	The current export flag
CURLEN	The current table line prefix length in a Huffman table
CURRANGELOW	The lower bound of the range of the current table line in a Huffman table
CURS	The current S coordinate in a text region decoding procedure
CURT	The current symbol instance's T coordinate relative to the current strip's T coordinate in a text region decoding procedure
CX	A label identifying an arithmetic coding context
D	Arithmetic coding decision
DFS	The difference in S coordinates between the first instances of two strips
DT	The number of empty strips between two non-empty strips
DW	The difference in width between two symbol bitmaps in a symbol dictionary decoding procedure
EXFLAGS	An array of export flags
EXINDEX	An index for the array EXFLAGS

EXRUNLENGTH	The length of a run of identical export flag values
EXTTEMPLATE	A parameter indicating whether extended reference template is used in a generic region decoding procedure
FIRSTS	The first S coordinate of the current strip
FIRSTCODE	The first code assigned to a particular prefix length in a Huffman table
GBATX₁	The X location of adaptive template pixel 1 in a generic region decoding procedure
GBATY₁	The Y location of adaptive template pixel 1 in a generic region decoding procedure
GBATX₂	The X location of adaptive template pixel 2 in a generic region decoding procedure
GBATY₂	The Y location of adaptive template pixel 2 in a generic region decoding procedure
GBATX₃	The X location of adaptive template pixel 3 in a generic region decoding procedure
GBATY₃	The Y location of adaptive template pixel 3 in a generic region decoding procedure
GBATX₄	The X location of adaptive template pixel 4 in a generic region decoding procedure
GBATY₄	The Y location of adaptive template pixel 4 in a generic region decoding procedure
GBATX₅	The X location of adaptive template pixel 5 in a generic region decoding procedure
GBATY₅	The Y location of adaptive template pixel 5 in a generic region decoding procedure
GBATX₆	The X location of adaptive template pixel 6 in a generic region decoding procedure
GBATY₆	The Y location of adaptive template pixel 6 in a generic region decoding procedure
GBATX₇	The X location of adaptive template pixel 7 in a generic region decoding procedure
GBATY₇	The Y location of adaptive template pixel 7 in a generic region decoding procedure
GBATX₈	The X location of adaptive template pixel 8 in a generic region decoding procedure
GBATY₈	The Y location of adaptive template pixel 8 in a generic region decoding procedure
GBATX₉	The X location of adaptive template pixel 9 in a generic region decoding procedure
GBATY₉	The Y location of adaptive template pixel 9 in a generic region decoding procedure
GBATX₁₀	The X location of adaptive template pixel 10 in a generic region decoding procedure
GBATY₁₀	The Y location of adaptive template pixel 10 in a generic region decoding procedure
GBATX₁₁	The X location of adaptive template pixel 11 in a generic region decoding procedure
GBATY₁₁	The Y location of adaptive template pixel 11 in a generic region decoding procedure
GBATX₁₂	The X location of adaptive template pixel 12 in a generic region decoding procedure
GBATY₁₂	The Y location of adaptive template pixel 12 in a generic region decoding procedure
GB	The prefix used for many of the variables associated with a generic (bitmap) region decoding procedure
GBCOLS	An array containing the colours used in a generic region segment
GBCOMBOP	The combination operator used in a generic region decoding procedure
GBFGCOLID	The 4-byte integer indicating the foreground colour of the generic region segment
GBH	The height of a generic region
GBREG	The region produced by a generic region decoding procedure
GBTEMPLATE	A parameter indicating the number and arrangement of the pixels in a template used in a generic region decoding procedure
GBW	The width of a generic region
GI	An array of gray-scale values

GR	The prefix used for many of the variables associated with a generic refinement region decoding procedure
GRATX₁	The X location of adaptive template pixel 1 in a generic refinement region decoding procedure
GRATY₁	The Y location of adaptive template pixel 1 in a generic refinement region decoding procedure
GRATX₂	The X location of adaptive template pixel 2 in a generic refinement region decoding procedure
GRATY₂	The Y location of adaptive template pixel 2 in a generic refinement region decoding procedure
GRAY	The current gray-scale value
GRAYMAX	The largest gray-scale value for which a pattern is given in a pattern dictionary decoding procedure
GRH	The height of a generic region being coded with refinement coding
GRREFERENCE	The reference bitmap in a generic refinement region decoding procedure
GRREFERENCEDX	The X offset of the reference bitmap with respect to the bitmap being decoded in a generic refinement region decoding procedure
GRREFERENCEDY	The Y offset of the reference bitmap with respect to the bitmap being decoded in a generic refinement region decoding procedure
GRREG	The region produced by a generic refinement region decoding procedure
GRTEMPLATE	A parameter indicating the number and arrangement of the pixels in a template used in decoding a generic region with refinement coding
GRW	The width of a generic region being coded with refinement coding
GS	The prefix used for many of the variables associated with a gray-scale image decoding procedure
GSBPP	The number of bits per gray-scale value in a gray-scale image decoding procedure
GSH	The height of the gray-scale image in a gray-scale image decoding procedure
GSKIP	A mask indicating gray-scale values to be skipped
GSMR	Whether MMR is used in a gray-scale image decoding procedure
GSTEMPLATE	A parameter indicating the number and arrangement of the pixels in a template used in a gray-scale image decoding procedure
GSUSESKIP	Whether some gray-scale values should be skipped in a gray-scale image decoding procedure
GSVALS	A decoded gray-scale image
GSW	The width of the gray-scale image in a gray-scale image decoding procedure
HB	The prefix used for many of the variables associated with a halftone (bitmap) region decoding procedure
HBH	The height of a halftone region
HBPP	The number of bits per value in an array of gray-scale values
HBW	The width of a halftone region
HCHEIGHT	The height of the current height class in a symbol dictionary decoding procedure
HCDH	The difference in height between two height classes in a symbol dictionary decoding procedure
HCFIRSTSYM	The index of the first symbol decoded in a height class
HCOMBOP	The combination operator used in a halftone region decoding procedure
HD	The prefix used for many of the variables associated with a pattern dictionary region decoding procedure
HDEFPIXEL	The default for pixels in a halftone region
HDMMR	Whether MMR is used in a pattern dictionary decoding procedure
HDPATS	Array of patterns produced by a pattern dictionary decoding procedure
HDPH	The height of the patterns in a pattern dictionary
HDPW	The width of the patterns in a pattern dictionary
HDTEMPLATE	The template identifier used to decode patterns in a pattern dictionary decoding procedure
HENABLESKIP	Whether unneeded gray-scale values are skipped in a halftone region decoding procedure

HGH	The height of the gray-scale image in a halftone region decoding procedure
HGW	The width of the gray-scale image in a halftone region decoding procedure
HGX	The horizontal offset of the grid in a halftone region decoding procedure
HGY	The vertical offset of the grid in a halftone region decoding procedure
H_I	The height of a symbol instance bitmap
HIGHPREFLEN	The prefix length of the upper range table line in a Huffman table
HMMR	Whether MMR coding is used in a halftone region decoding procedure
HNUMPATS	The number of patterns that may be used in a halftone region decoding procedure
HO_I	The height of the original bitmap of a symbol instance containing refinement information
HPATS	Array of patterns used in a halftone region
HPH	The height of each pattern in a halftone region
HPW	The width of each pattern in a halftone region
HRX	The horizontal coordinate of a halftone grid vector
HRY	The vertical coordinate of a halftone grid vector
HSKIP	A mask indicating gray-scale values to be skipped
HT	The prefix used for many of the variables associated with a Huffman table decoding procedure
HTEMPLATE	A parameter indicating the number and arrangement of the pixels in a template used in a halftone region decoding procedure
HTHIGH	One greater than the largest value that is represented by any normal table line in a Huffman table
HTLOW	The lowest value that is represented by any normal table line in a Huffman table
HTOFFSET	The range offset of a table line when decoding using a Huffman table
HTOOB	Whether a Huffman table can produce the out-of-band value OOB
HTPS	The length of the encoded prefix field in a table line in a Huffman table
HTREG	The region produced by a halftone region decoding procedure
HTRS	The length of the encoded range field in a table line in a Huffman table
HTVAL	The value decoded using a Huffman table rappeller
I	The array, indexed by CX, of the indices of the adaptive probability estimates
I	An array index
IAAI	An arithmetic integer decoding procedure used to decode the number of symbol instances in an aggregation
IADH	An arithmetic integer decoding procedure used to decode the difference in height between two height classes
IADS	An arithmetic integer decoding procedure used to decode the S coordinate of the second and subsequent symbol instances in a strip
IADT	An arithmetic integer decoding procedure used to decode the T coordinate of the second and subsequent symbol instances in a strip
IADW	An arithmetic integer decoding procedure used to decode the difference in delta width between two symbols in a height class
IAEX	An arithmetic integer decoding procedure used to decode export flags
IAFS	An arithmetic integer decoding procedure used to decode the S coordinate of the first symbol instance in a strip
IAID	An arithmetic integer decoding procedure used to decode the symbol IDs of symbol instances
IARDH	An arithmetic integer decoding procedure used to decode the delta height of symbol instance refinements
IARDW	An arithmetic integer decoding procedure used to decode the delta width of symbol instance refinements
IARDX	An arithmetic integer decoding procedure used to decode the delta X values of symbol instance refinements

IARDY	An arithmetic integer decoding procedure used to decode the delta Y values of symbol instance refinements
IARI	An arithmetic integer decoding procedure used to decode the R_I bit of symbol instances
IAIT	An arithmetic integer decoding procedure used to decode the T coordinate of the symbol instances in a strip
IB_I	The bitmap of a symbol instance
IBO_I	The original bitmap of a symbol instance containing refinement information
ID_I	The symbol ID of a symbol instance
IDS	The delta S value for a symbol instance in a text region decoding procedure
J	An array index
K	The ordinal for a referred-to segment
LENCOUNT	A histogram of the prefix lengths in a Huffman table
LENMAX	The largest prefix length in a Huffman table
LOGSBSTRIPS	The base-2 logarithm of the strip size used to encode a text region
LOWPREFLEN	The prefix length of the lower range table line in a Huffman table
LTP	Whether the current line is coded explicitly in a generic region decoding procedure or a generic refinement region decoding procedure
m_g	Horizontal index for the current gray-scale value
MMR	Whether MMR coding is used in a generic region decoding procedure
MPS	The array, indexed by CX, of the current more probable binary values
NINSTANCES	A symbol instance counter
n_g	Vertical index for the current gray-scale value
NLPS	The next index for an LPS renormalization
NMPS	The next index for an MPS renormalization
NSYMSDECODED	The number of symbols decoded so far in a symbol dictionary decoding procedure
NTEMP	The number of table lines in a Huffman table
OOB	An out-of-band value
P	The page with which a segment is associated
PREFLEN	An array of prefix lengths representing the table lines in a Huffman table
Q_e	An estimate of the LPS probability
r	A segment retention flag
R	The number of segments referred to by some segment
RANGELEN	An array of the lengths of the ranges of the table lines in a Huffman table
RANGELOW	An array of the lower bounds of the ranges of the table lines in a Huffman table
RA_1, RA_2	Adaptive template pixels in the generic refinement region decoding procedure
RDH_I	The delta height of a symbol instance refinement bitmap
RDW_I	The delta width of a symbol instance refinement bitmap
RDX_I	The X offset of a symbol instance refinement
RDY_I	The Y offset of a symbol instance refinement
REFAGGNINST	The number of symbol instances in an aggregation
R_I	A bit indicating whether refinement information is present for a symbol instance
REFCORNER	Which corner of a symbol instance bitmap is to be used as a reference in a text region decoding procedure
S	One coordinate of the coordinate system used in a text region decoding procedure
S_I	The S coordinate of a symbol instance

SB	The prefix used for many of the variables associated with a symbol (bitmap) region decoding procedure
SBCOLS	An array of colours used in a text region segment
SBCOLSECTSIZE	The colour section length
SBDISOFFSET	An offset for the coded delta S values in a text region
SBCOMBOP	The combination operator used in a text region decoding procedure
SBDEFPIXEL	The default for pixels in a text region
SBFGCOLID	The three 1-byte integers indicating the foreground colour of the text region segment
SBH	The height of a text region
SBHUFF	Whether Huffman coding is used in a text region decoding procedure
SBHUFFDS	The Huffman table used to decode the S coordinate of subsequent symbol instances in a strip
SBHUFFDT	The Huffman table used to decode the difference in T coordinates between non-empty strips
SBHUFFFS	The Huffman table used to decode the S coordinate of the first symbol instance in a strip
SBHUFFRDH	The Huffman table used to decode the difference between a symbol's height and the height of a refinement coded symbol instance bitmap
SBHUFFRDW	The Huffman table used to decode the difference between a symbol's width and the width of a refinement coded symbol instance bitmap
SBHUFFRDX	The Huffman table used to decode the difference between a symbol instance's X coordinate and the X coordinate of a refinement coded bitmap
SBHUFFRDY	The Huffman table used to decode the difference between a symbol instance's Y coordinate and the Y coordinate of a refinement coded symbol instance bitmap
SBHUFFRSIZE	The Huffman table used to decode the size of a symbol instance's refinement bitmap data
SBNUMINSTANCES	The number of symbol instances in a text region
SBNUMSYMS	The number of symbols that may be used in a text region
SBRTX₁	The X location of the adaptive template pixel RA ₁ in a text region decoding procedure
SBRTY₁	The Y location of the adaptive template pixel RA ₁ in a text region decoding procedure
SBRTX₂	The X location of the adaptive template pixel RA ₂ in a text region decoding procedure
SBRTY₂	The Y location of the adaptive template pixel RA ₂ in a text region decoding procedure
SBREFINE	Whether refinement coding is used in a text region decoding procedure
SBREG	The region produced by a text region decoding procedure
SBRTEMPLATE	Template identifier for refinement coding of bitmap in a text region decoding procedure
SBSTRIPS	The height of the symbol instance strips
SBSYMCODELEN	The length of the symbol codes used in IAID
SBSYMCODES	An array of variable-length codes identifying individual symbols
SBSYMS	An array of symbols used in a text region
SBW	The width of a text region
SD	The prefix used for many of the variables associated with a symbol dictionary region decoding procedure
SDATX₁	The X location of the adaptive template pixel A ₁ in a symbol dictionary decoding procedure
SDATY₁	The Y location of the adaptive template pixel A ₁ in a symbol dictionary decoding procedure
SDATX₂	The X location of the adaptive template pixel A ₂ in a symbol dictionary decoding procedure
SDATY₂	The Y location of the adaptive template pixel A ₂ in a symbol dictionary decoding procedure
SDATX₃	The X location of the adaptive template pixel A ₃ in a symbol dictionary decoding procedure
SDATY₃	The Y location of the adaptive template pixel A ₃ in a symbol dictionary decoding procedure
SDATX₄	The X location of the adaptive template pixel A ₄ in a symbol dictionary decoding procedure
SDATY₄	The Y location of the adaptive template pixel A ₄ in a symbol dictionary decoding procedure

SDEXSyms	The symbols exported from a symbol dictionary
SDHUFF	Whether Huffman coding is used in a symbol dictionary decoding procedure
SDHUFFAGGINST	The Huffman table used to decode the number of symbol instances in an aggregation in a symbol dictionary decoding procedure
SDHUFFDH	The Huffman table used to decode the difference in height between two height classes in a symbol dictionary decoding procedure
SDHUFFDW	The Huffman table used to decode the difference in width between two symbols in a symbol dictionary decoding procedure
SDHUFFBMSIZE	The Huffman table used to decode the size of a height class collective bitmap in a symbol dictionary decoding procedure
SDINSYMS	An array of symbols used as a parameter to a symbol dictionary decoding procedure
SDNEWSYMS	The symbols decoded in a symbol dictionary
SDNEWSYMWIDTHS	The widths of the symbols decoded in a symbol dictionary
SDNUMEXSYMS	The number of symbols exported from a symbol dictionary
SDNUMINSYMS	The number of symbols in the array that is used as a parameter to a symbol dictionary decoding procedure
SDNUMNEWSYMS	The number of symbols generated in a symbol dictionary
SDREFAGG	Whether refinement and aggregate coding are used in a symbol dictionary decoding procedure
SDRATX₁	The X location of the adaptive template pixel RA ₁ in a symbol dictionary decoding procedure
SDRATY₁	The Y location of the adaptive template pixel RA ₁ in a symbol dictionary decoding procedure
SDRATX₂	The X location of the adaptive template pixel RA ₂ in a symbol dictionary decoding procedure
SDRATY₂	The Y location of the adaptive template pixel RA ₂ in a symbol dictionary decoding procedure
SDRTEMPLATE	Template identifier for refinement coding of bitmaps in a symbol dictionary decoding procedure
SDTEMPLATE	The template identifier used to decode symbol bitmaps in a symbol dictionary decoding procedure
SKIP	A mask of pixels to be skipped during the decoding of a generic region
SLTP	A binary value indicating whether the current line was typically predicted and the previous line was not, or vice versa
STRIPT	The numerically smallest T coordinate in the current strip
SWITCH	Whether MPS and LPS are switched on an LPS renormalization
SYMWIDTH	The current bitmap width in a symbol dictionary decoding procedure
T	One coordinate of the coordinate system used in a text region decoding procedure
TEMPC	A temporary register in the MQ coder
T_I	The T coordinate of a symbol instance
TOTWIDTH	The total width of the bitmaps in a height class
TPGDON	Whether typical prediction is used in a generic region decoding procedure
TPGRON	Whether typical prediction is used in a generic region refinement decoding procedure
TPGRPIX	Whether the current pixel is to be decoded implicitly using a TPGR prediction
TPGRVAL	The value of the TPGR-predicted current pixel
TRANPOSED	Whether the symbol instance coordinates are transposed in a text region decoding procedure
USESKIP	Whether some pixels should be skipped in the decoding of a generic region
V1	A binary value
V2	A binary value
W_I	The width of a symbol instance bitmap
WO_I	The width of the original bitmap of a symbol instance containing refinement information
x	The horizontal coordinate of a location on a halftone grid

X	The horizontal coordinate of a pixel in a bitmap
y	The vertical coordinate of a location on a halftone grid
Y	The vertical coordinate of a pixel in a bitmap

4.3 Operator definitions

The following operators are defined:

OR	If V1 and V2 are two binary values, then V1 OR V2 is equal to 0 if both V1 and V2 are 0 . It is equal to 1 if either of V1 or V2 is 1 . If V1 and V2 are two integer values, then it is the result of bitwise application of OR.
AND	If V1 and V2 are two binary values, then V1 AND V2 is equal to 0 if either of V1 or V2 is 0 . It is equal to 1 if both V1 and V2 are 1 . If V1 and V2 are two integer values, then it is the result of bitwise application of AND.
XOR	If V1 and V2 are two binary values, then V1 XOR V2 is equal to 0 if V1 and V2 are equal. It is equal to 1 if V1 and V2 differ. If V1 and V2 are two integer values, then it is the result of bitwise application of XOR.
XNOR	If V1 and V2 are two binary values, then V1 XNOR V2 is equal to 0 if V1 and V2 differ. It is equal to 1 if V1 and V2 are equal.
REPLACE	If V1 and V2 are two binary values, then V1 REPLACE V2 is equal to V2.
NOT	If V1 is a binary value, then NOT V1 is 1 if V1 is 0 , and is 0 if V1 is 1 .
min	If x and y are numbers, then min(x, y) is the smaller of x and y.
max	If x and y are numbers, then max(x, y) is the larger of x and y.
$\lfloor a \rfloor$	If a is a number, then $\lfloor a \rfloor$ is the largest integer less than or equal to a.
$\lceil a \rceil$	If a is a number, then $\lceil a \rceil$ is the smallest integer greater than or equal to a.
<<	If V1 and V2 are two integers, then V1 << V2 is the value obtained by shifting the value of V1 leftwards by V2 bits, filling the rightmost V2 bits of the new value with 0 .
>>	If V1 and V2 are two integers, then V1 >> V2 is the value obtained by shifting the value of V1 rightward by V2 bits, filling the leftmost V2 bits of the new value with 0 .
>> _A	If V1 and V2 are two integers, then V1 >> _A V2 is the value obtained by shifting the value of V1 rightward by V2 bits, filling the leftmost V2 bits of the new value with 0 if V1 is non-negative and 1 if V1 is negative.

5 Conventions

5.1 Typographic conventions

All parameter names are given in **bold face**.

5.2 Binary notation

The two binary values are denoted as **0** and **1**.

5.3 Hexadecimal notation

The prefix 0x indicates that the following value is to be interpreted as a hexadecimal number (radix 16).

EXAMPLE – The value 0x6A is equal to the decimal value 106.

5.4 Integer value syntax

5.4.1 Bit packing

Bits are packed into bytes starting at the most significant bit. If a decoder is reading a sequence of bits out of a bitstream, it shall first read the most significant bit of the first byte, then the next most significant bit, and so on, then proceed to the next byte.

EXAMPLE – The sequence of bytes 0x2F 0x05 0xC1, if interpreted as a sequence of bits, is the sequence

0 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1

5.4.2 Multi-byte values

All multi-byte values shall be interpreted in a most-significant-first manner: the first byte of each value is the most significant, and the last byte is the least significant.

EXAMPLE – The sequence of bytes 0x01 0x5C 0x99 0xFA, if interpreted as a four-byte value, represents the value 0x015C99FA.

5.4.3 Bit numbering

The least significant bit of any value is numbered bit 0. For a one-byte value, the most significant bit is numbered bit 7; for a two-byte value, the most significant bit is numbered bit 15; for a four-byte value, the most significant bit is numbered bit 31.

5.4.4 Signedness

Unless otherwise specified, all multi-bit values shall be treated as unsigned values. When a value is to be treated as a signed number, it shall be interpreted in two's-complement form.

5.5 Array notation and conventions

Arrays are numbered starting from zero.

EXAMPLE – A one-dimensional array ARR containing twelve elements has elements:

ARR[0], ARR[1], . . . , ARR[11]

5.6 Image and bitmap conventions

A bitmap is a rectangular array. Every element in this array has the value **0** or **1**. An element in a bitmap is referred to as a pixel.

NOTE 1 – Throughout this Recommendation | International Standard, pixels in bitmaps are treated as having the values **0** or **1**. In most applications of this Recommendation | International Standard, the application will select some interpretation of these two values. A typical interpretation of these pixels is that **0** represents white, or background, and **1** represents black, or foreground. However, this is not a requirement of this Recommendation | International Standard and applications are free to make other interpretations of these values.

The terms "left", "right", "top", "bottom", "width" and "height" are often applied to bitmaps. These terms do not refer to any physical aspect of the bitmap: if a bitmap is printed on paper, it may be printed with its "left" edge along any edge of the paper. They are used within this Recommendation | International Standard to refer to the four edges of the bitmap as shown in Figure 2.

A pixel in a bitmap is referred to by a pair of coordinates X and Y, sometimes written as a pair (X, Y). The location (0, 0) represents the pixel in the top left corner. The X coordinate increases rightwards and the Y coordinate increases downwards.

If *BM* is a bitmap, then the pixel whose coordinates are X and Y is referred to as *BM*[X, Y].

NOTE 2 – These conventions are intended to make it easier to describe operations involving bitmaps, and are not intended to imply any physical characteristics of the image represented by the bitmap.

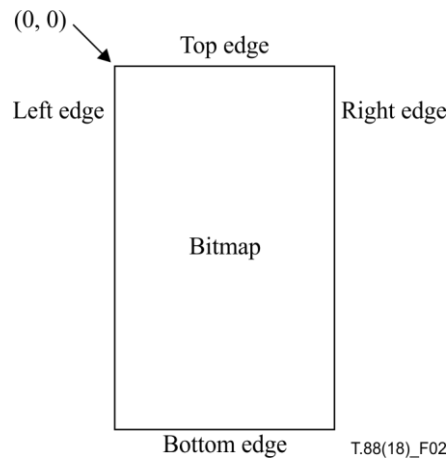


Figure 2 – The four edges of a bitmap

6 Decoding Procedures

6.1 Introduction to decoding procedures

This Recommendation | International Standard makes use of a number of different decoding procedures for different types of data. Each of these decoding procedures produces a certain kind of data as output. The generic region decoding procedure, generic refinement region decoding procedure, halftone region decoding procedure, and text region decoding procedures all produce regions as their output. The symbol dictionary decoding procedure produces an array of symbols as its output. The pattern dictionary decoding procedure similarly produces an array of halftone cell bitmaps as its output.

The various region decoding procedures operate in different ways:

- The generic region decoding procedure decodes a bitmap, treating it simply as an array of binary pixels.
- The generic region refinement decoding procedure decodes a bitmap by treating it as an array of binary pixels, but coding each pixel with respect to some reference bitmap.
- The text region decoding procedure decodes a bitmap by drawing a collection of symbols into it, possibly applying the generic refinement region decoding procedure to each one.
- The halftone region decoding procedure decodes a bitmap by placing a collection of patterns into it, at locations specified by a halftone grid.

Each decoding procedure is specified in terms of a number of parameters and a sequence of operations, which are affected by the values of the parameters. These parameters are supplied to the decoding procedure for each invocation, and the same decoding procedure may be invoked multiple times during the course of decoding a bitstream, with different parameters each time.

Some of the decoding procedure parameters are unused in certain circumstances, usually depending on the values of other parameters. In these circumstances, no value needs to be specified for those unused parameters.

In this clause, subsequent clauses, and normative annexes, restrictions are placed on the bitstream being decoded.

EXAMPLE 1 – In 7.3, some segment types are described as "Reserved; must not be used."

EXAMPLE 2 – In 7.4.2.1.1, if the **SDHUFF** field is **0** then the **SDHUFFDH** field must contain the value **0**.

These restrictions should be interpreted as meaning that the behaviour of a decoder encountering a bitstream that does not satisfy the restrictions is undefined, and is outside the scope of this Recommendation | International Standard.

NOTE – This means that if a decoder encounters a bitstream that does not satisfy the restrictions, it may take any action: it may give up and abort decoding; it may ignore the error and attempt to continue; it may interpret the error and change its behaviour (e.g., use the error to attempt to aid recovery from further errors); and so on.

6.2 Generic region decoding procedure

6.2.1 General description

This decoding procedure is used to decode a rectangular array of **0** or **1** values, which are coded one pixel at a time (i.e., it is used to decode a bitmap using simple, generic, coding). The decoding procedure also modifies an array of probability information which may be used by other invocations of this generic region decoding procedure.

The generic region decoding procedure may be based on sequential coding of the image pixels using arithmetic coding as specified in Annex E and a template to determine the coding state. This technique was used in Rec. ITU-T T.82 | ISO/IEC 11544 (JBIG). This type of decoding is described in 6.2.5.

Alternatively, for improved speed but reduced compression the generic region decoding procedure may be based on Huffman coding of runs of pixels. This technique was used in the MMR (Modified Modified READ) algorithm described in Recommendation ITU-T T.6 (G4). This type of decoding is described in 6.2.6.

6.2.2 Input parameters

The parameters to this decoding procedure are shown in Table 2.

Table 2 – Parameters for the generic region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
MMR	Integer	1	N	Whether MMR coding is used.
GBW	Integer	32	N	The width of the region.
GBH	Integer	32	N	The height of the region.
GBTEMPLATE	Integer	2	N	The template identifier. ^{a)}
TPGDON	Integer	1	N	Whether typical prediction is used. ^{a)}
EXTTEMPLATE	Integer	1	N	Whether extended reference template is used. ^{e)}
USESKIP	Integer	<u>1</u>	<u>N</u>	Whether some pixels should be skipped in the decoding. ^{a)}
SKIP	Bitmap			A bitmap indicating which pixels should be skipped. GBW pixels wide, GBH pixels high. ^{c)}
GBATX₁	Integer	8	Y	The X location of the adaptive template pixel A ₁ . ^{a)}
GBATY₁	Integer	8	Y	The Y location of the adaptive template pixel A ₁ . ^{a)}
GBATX₂	Integer	8	Y	The X location of the adaptive template pixel A ₂ . ^{b)}
GBATY₂	Integer	8	Y	The Y location of the adaptive template pixel A ₂ . ^{b)}
GBATX₃	Integer	8	Y	The X location of the adaptive template pixel A ₃ . ^{b)}
GBATY₃	Integer	8	Y	The Y location of the adaptive template pixel A ₃ . ^{b)}
GBATX₄	Integer	8	Y	The X location of the adaptive template pixel A ₄ . ^{b)}
GBATY₄	Integer	8	Y	The Y location of the adaptive template pixel A ₄ . ^{b)}
GBATX₅	Integer	8	Y	The X location of the adaptive template pixel A ₅ . ^{d)}
GBATY₅	Integer	8	Y	The Y location of the adaptive template pixel A ₅ . ^{d)}
GBATX₆	Integer	8	Y	The X location of the adaptive template pixel A ₆ . ^{d)}
GBATY₆	Integer	8	Y	The Y location of the adaptive template pixel A ₆ . ^{d)}
GBATX₇	Integer	8	Y	The X location of the adaptive template pixel A ₇ . ^{d)}
GBATY₇	Integer	8	Y	The Y location of the adaptive template pixel A ₇ . ^{d)}
GBATX₈	Integer	8	Y	The X location of the adaptive template pixel A ₈ . ^{d)}
GBATY₈	Integer	8	Y	The Y location of the adaptive template pixel A ₈ . ^{d)}
GBATX₉	Integer	8	Y	The X location of the adaptive template pixel A ₉ . ^{d)}
GBATY₉	Integer	8	Y	The Y location of the adaptive template pixel A ₉ . ^{d)}
GBATX₁₀	Integer	8	Y	The X location of the adaptive template pixel A ₁₀ . ^{d)}
GBATY₁₀	Integer	8	Y	The Y location of the adaptive template pixel A ₁₀ . ^{d)}
GBATX₁₁	Integer	8	Y	The X location of the adaptive template pixel A ₁₁ . ^{d)}
GBATY₁₁	Integer	8	Y	The Y location of the adaptive template pixel A ₁₁ . ^{d)}

Table 2 – Parameters for the generic region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
GBATX₁₂	Integer	8	Y	The X location of the adaptive template pixel A ₁₂ . ^{d)}
GBATY₁₂	Integer	8	Y	The Y location of the adaptive template pixel A ₁₂ . ^{d)}
GBCOLS	Array of colours (colour palette)			An array containing the colours used in this generic region. ^{g)}
GBCOMBOP	Operator			The combination operator for this generic region. Shall take on the value REPLACE. ^{g)}
COLEXTFLAG	Integer	1	N	A parameter indicating whether the generic region segment is extended to represent coloured bitmap.
a) Unused if MMR = 1 b) Unused if MMR = 1 or GBTEMPLATE ≠ 0 c) Unused if USESKIP = 0 or MMR = 1 d) Used only if MMR = 0 and GBTEMPLATE = 0 and EXTTEMPLATE = 1 e) Used only if MMR = 0 and GBTEMPLATE = 0 f) Unused if COLEXTFLAG = 0.				

6.2.3 Return value

The variable whose value is the result of this decoding procedure is shown in Table 3.

Table 3 – Return value from the generic region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
GBREG	Bitmap			The decoded region bitmap.

6.2.4 Variables used in decoding

The variables used by this decoding procedure are shown in Table 4.

Table 4 – Variables used in the generic region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
LTP	Integer	1	N	Whether the current image line is coded explicitly. ^{a)}
SLTP	Integer	1	N	Whether the current line's LTP value differs from the previous line's LTP value. ^{a)}
CONTEXT	Integer	16	N	The values of the pixels in the template. ^{a)}
GBFGCOLID	Integer	32	N	The 4-byte integer indicating the colour palette ID of the foreground colour for the generic region segment. ^{b)}
a) Unused if MMR =1 b) Unused if COLEXTFLAG = 0.				

6.2.5 Decoding using a template and arithmetic coding

6.2.5.1 General description

If **MMR** is 0 the generic region decoding procedure is based on arithmetic coding with a template to determine the coding state. The remainder of 6.2.5 describes this form of decoding, and only applies when **MMR** is 0.

6.2.5.2 Coding order and edge conventions

The coding algorithm iterates through the bitmap in raster scan order, that is, by rows from top to bottom, and within each row from left to right. The processing for a current target pixel will reference some pixels in fixed spatial relationship to the target pixel.

Near the edges of the bitmap, these neighbour references might not lie in the actual bitmap. The rule to satisfy out-of-bounds references shall be:

- All pixels lying outside the bounds of the actual bitmap have the value **0**.

6.2.5.3 Fixed templates

A template defines a neighbourhood around a pixel to be coded. The values of the pixels in this neighbourhood define a context. Each context has its own adaptive probability estimate used by the arithmetic coder (see Annex E). Although a template is a geometric pattern of pixels, the pixels in a template are said to take on values when the template is aligned with a particular part of the image.

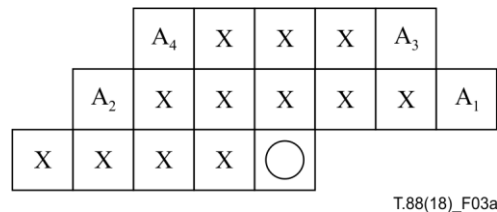


Figure 3(a) – Template when GBTEMPLATE = 0 and EXTTEMPLATE = 0, showing the AT pixels at their nominal locations

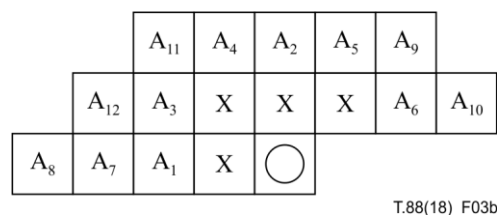


Figure 3(b) – Template when GBTEMPLATE = 0 and EXTTEMPLATE = 1, showing the AT pixels at their nominal locations

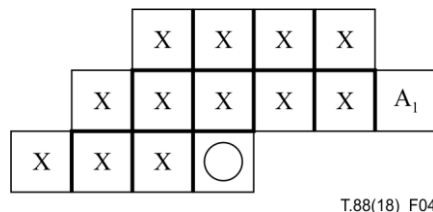


Figure 4 – Template when GBTEMPLATE = 1 showing the AT pixels at its nominal location

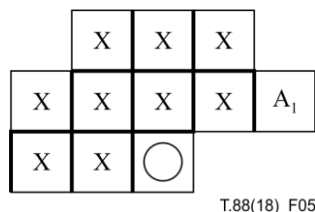


Figure 5 – Template when GBTEMPLATE = 2 showing the AT pixels at its nominal location

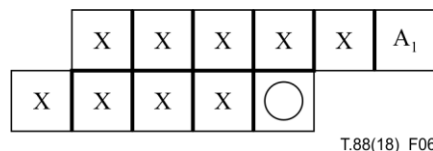


Figure 6 – Template when GBTEMPLATE = 3 showing the AT pixels at its nominal location

NOTE 1 – A rule of thumb is to use large templates for large bitmaps. Thus a full-size periodic halftone should be coded with the 16-pixel template and tiny bitmaps such as usual symbol bitmaps should be coded with one of the 10-pixel templates. In some cases an intermediate template is desired, for performance or decoder memory requirements; in this case the 13-pixel template should be used. It is also possible to generate further templates by placing one or more of the AT pixels on top of a regular template pixel, thus fixing its value.

NOTE 2 – The 10-pixel templates are those used in Rec. ITU-T T.82 | ISO/IEC 11544 (JBIG). Software execution speed is somewhat higher with the two-line template than any of the three-line templates. For most images the 10-pixel, three-line template gives higher compression than the 10-pixel, two-line template.

6.2.5.4 Adaptive template pixels

In coding the image, the template shall be allowed to change in the restricted way described in this clause.

The pixels that are allowed to change are called AT pixels. Their nominal locations are indicated by 'A₁', 'A₂', 'A₃', 'A₄', 'A₅', 'A₆', 'A₇', 'A₈', 'A₉', 'A₁₀', 'A₁₁', and 'A₁₂' in Figures 3(a), 3(b), 4, 5 and 6. Note that some templates have fewer than four AT pixels. In general, an AT pixel can be located anywhere in the field shown in Figure 7, not including the current pixel. Hence, there is the possibility of using an effective template size of 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, or 4 pixels by having the moved location of the AT pixel overlap a regular template pixel. The actual locations of the AT pixels for any invocation of this decoding procedure are specified as parameters to the decoding procedure. The location of the pixel A₁ is given by (GBATX₁, GBATY₁). If GBTEMPLATE is 0 then:

- the location of the pixel A₂ is given by (GBATX₂, GBATY₂);
- the location of the pixel A₃ is given by (GBATX₃, GBATY₃);
- and the location of the pixel A₄ is given by (GBATX₄, GBATY₄).

Additionally, if GBTEMPLATE is 0 and EXTTEMPLATE is 1, then:

- the location of the pixel A₅ is given by (GBATX₅, GBATY₅);
- the location of the pixel A₆ is given by (GBATX₆, GBATY₆);
- the location of the pixel A₇ is given by (GBATX₇, GBATY₇);
- the location of the pixel A₈ is given by (GBATX₈, GBATY₈);
- the location of the pixel A₉ is given by (GBATX₉, GBATY₉);
- the location of the pixel A₁₀ is given by (GBATX₁₀, GBATY₁₀);
- the location of the pixel A₁₁ is given by (GBATX₁₁, GBATY₁₁);
- the location of the pixel A₁₂ is given by (GBATX₁₂, GBATY₁₂).

NOTE 1 – Some profiles may restrict AT pixel locations to a smaller range than that shown in Figure 7.

NOTE 2 – The indices of the AT pixels in Figures 3(a) and 3(b) correspond to the expected goodness. If moving only one AT pixel from its nominal location shown in Figure 3(a), it is advisable to move A₄. The next pixel to move is A₃ and so on.

NOTE 3 – The nominal locations of the AT pixels are as shown in Table 5. These locations should be used unless other locations improve compression performance. Some profiles may restrict AT pixel locations to only these nominal locations.

NOTE 4 – If an AT pixel's location overlaps any regular template pixel's location, then the AT pixel's value can be ignored (since it duplicates another value). This can reduce the memory requirements of the decoder, since not all CX values can occur. However, when TPGD is enabled (TPGDON = 1), the context used to code the SLTP value is used, regardless of whether AT pixels overlap regular template pixels. This means that contexts where the AT pixel's value differs from the regular template pixel's value can still occur, but only for SLTP when TPGD is enabled.

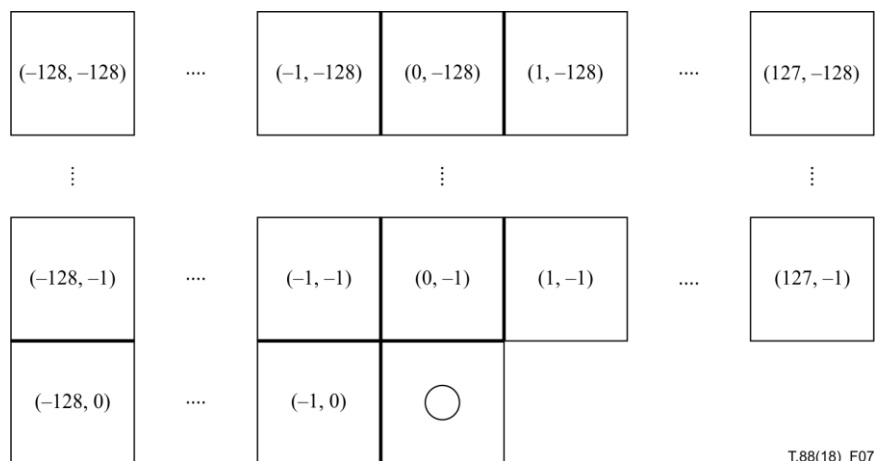


Figure 7 – Field to which AT pixel locations are restricted

Table 5 – The nominal values of the AT pixel locations

GBTEMPLATE		<u>0</u>		0		1		2		3	
EXTTEMPLATE		1		0		0		0		0	
GBATX ₁	GBATY ₁	–2	<u>0</u>	3	–1	3	–1	2	–1	2	–1
GBATX ₂	GBATY ₂	0	–2	–3	–1	NA	NA	NA	NA	NA	NA
GBATX ₃	GBATY ₃	–2	–1	2	–2	NA	NA	NA	NA	NA	NA
GBATX ₄	GBATY ₄	–1	–2	–2	–2	NA	NA	NA	NA	NA	NA
GBATX ₅	GBATY ₅	1	–2	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₆	GBATY ₆	2	–1	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₇	GBATY ₇	–3	0	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₈	GBATY ₈	–4	0	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₉	GBATY ₉	2	–2	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₁₀	GBATY ₁₀	3	–1	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₁₁	GBATY ₁₁	–2	–2	NA	NA	NA	NA	NA	NA	NA	NA
GBATX ₁₂	GBATY ₁₂	–3	–1	NA	NA	NA	NA	NA	NA	NA	NA

NOTE – NA means that the parameter has no nominal value.

6.2.5.5 Typical prediction for generic direct coding (TPGD)

Typical prediction for generic direct coding can be enabled or disabled with the **TPGDON** parameter. If typical prediction for generic direct coding is enabled (**TPGDON** is **1**), then before the first pixel of each row is decoded, a value indicating that a row is typical shall be decoded. If the row is typical then each pixel of this row is identical to the corresponding pixel in the row immediately above, and so no other pixels of this row need to be decoded. If the row is not typical, then each pixel of this row needs to be decoded.

NOTE – The value decoded before the first pixel of each row is not used in any pixel's template.

6.2.5.6 Skipped pixels

If the parameter **USESKIP** is **1**, then the parameter **SKIP** contains a **GBW**-by-**GBH** bitmap. Each pixel in **SKIP** corresponds to a pixel in the bitmap being decoded; if the pixel in **SKIP** is **1** then the corresponding pixel in the bitmap being decoded is **0** and is not actually decoded.

6.2.5.7 Decoding the bitmap

The decoding of the bitmap proceeds as follows:

- 1) Set:

$$LTP = 0$$

- 2) Create a bitmap GBREG of width **GBW** and height **GBH** pixels.
- 3) Decode each row as follows:
 - a) If all **GBH** rows have been decoded then the decoding is complete; proceed to step 4).
 - b) If **TPGDON** is **1**, then decode a bit using the arithmetic entropy coder, where the context used to decode this bit varies depending on the template in use:
 - If **GBTEMPLATE** is 0, use the context shown in Figure 8.
 - If **GBTEMPLATE** is 1, use the context shown in Figure 9.
 - If **GBTEMPLATE** is 2, use the context shown in Figure 10.
 - If **GBTEMPLATE** is 3, use the context shown in Figure 11.

Let SLTP be the value of this bit. Set:

$$LTP = LTP \text{ XOR } SLTP$$

NOTE – In Figures 8 to 11, the template is shown with the AT pixel or pixels in their nominal locations. The same pixel values (**0** or **1**) shall be used for the AT pixels no matter what their actual locations are. That is, moving the AT pixels does not affect the context that is used to decode SLTP.

- c) If **LTP = 1** then set every pixel of the current row of **GBREG** equal to the corresponding pixel of the row immediately above.
- d) If **LTP = 0** then, from left to right, decode each pixel of the current row of **GBREG**. The procedure for each pixel is as follows:
- If **USESKIP** is **1** and the pixel in the bitmap **SKIP** at the location corresponding to the current pixel is **1**, then set the current pixel to **0**.
 - Otherwise:
- Place the template given by parameters **GBTEMPLATE**, **GBATX₁** through **GBATX₁₂** and **GBATY₁** through **GBATY₁₂** so that the current pixel is aligned with the location denoted by a circle in the figure describing the appearance of the template with identifier **GBTEMPLATE**.
 - Form an integer **CONTEXT** by gathering the values of the image pixels overlaid by the template (including **AT** pixels) at its current location. The order of this gathering is not standardized, but shall be consistent and independent of the location of the **AT** pixels.
 - Decode the current pixel by invoking the arithmetic entropy decoding procedure, with **CX** set to the value formed by concatenating the label "GB" and the 10-16 pixel values gathered in **CONTEXT**. The result of this invocation is the value of the current pixel.
- EXAMPLE – If **GBTEMPLATE** is 2, the image pixels overlaid by the template are as shown in Figure 10, and the pixels are gathered in reading order (in rows from top to bottom, and within each row from left to right), then **CX** is set to "GB0011100101".
- 4) After all the rows have been decoded, the current contents of the bitmap **GBREG** are the results that shall be obtained by every decoder, whether it performs this exact sequence of steps or not.

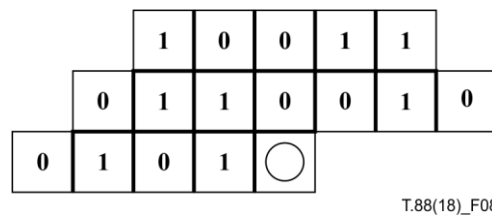


**Figure 8 – Reused context for coding the SLTP value
when **GBTEMPLATE** is 0**

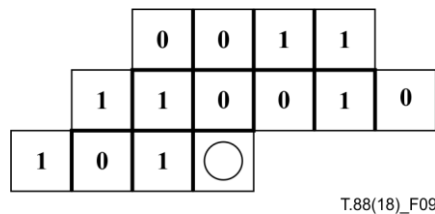


**Figure 9 – Reused context for coding the SLTP value
when **GBTEMPLATE** is 1**

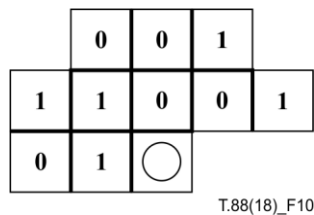


**Figure 10 – Reused context for coding the SLTP value
when **GBTEMPLATE** is 2**

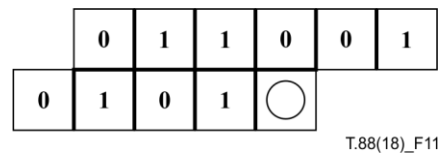


**Figure 11 – Reused context for coding the SLTP value
when GBTEMPLATE is 3**

6.2.6 Decoding using MMR coding

If **MMR** is **1**, the generic region decoding procedure is identical to an MMR (Modified Modified READ) decoder described in Recommendation ITU-T T.6, with the following exceptions:

- An invocation of the generic region decoding procedure with **MMR** equal to **1** shall consume an integral number of bytes, beginning and ending on a byte boundary. This may involve skipping over some bits in the last byte read.
- The decoder in Recommendation ITU-T T.6 is specified as producing pixels whose value may be either "black" or "white". For the purposes of this Recommendation | International Standard, the result of using the MMR decoder shall be interpreted as follows:
 - Pixels decoded by the MMR decoder having the value "black" shall be treated as having the value **1**.
 - Pixels decoded by the MMR decoder having the value "white" shall be treated as having the value **0**.
- If the number of bytes contained in the encoded bitmap is known in advance, then it is permissible for the data stream not to contain an EOFB (**000000000001000000000001**) at the end of the MMR-encoded data. The cases where the number of bytes is known are when this decoding procedure is invoked:
 - from within the pattern dictionary decoding procedure;
 - from within the symbol dictionary decoding procedure; or
 - as part of decoding a generic region whose data length is known.

The number of bytes is not known when this decoding procedure is invoked from within the gray-scale image decoding procedure, or when it is invoked as part of a decoding generic region whose data length is not known. In these cases, EOFB must be present.

NOTE 1 – The sources of the byte count, in the cases where it is known, are:

- Within the pattern dictionary decoding procedure, the byte count is known because all the segment data, beyond the fixed-length data header, is a single MMR-encoded data block, so the MMR data length can be computed from the segment data length.
- Within the symbol dictionary decoding procedure, the byte count is known from BMSIZE.
- Within the generic region decoding procedure, the byte count is again known from the segment data length.

The reason for allowing EOFB to be optional is that an EOFB is three bytes long, while the byte count is often known beforehand, or can be encoded in fewer than three bytes. Thus, omitting an EOFB reduces the bitmap's coded data size; in symbol dictionaries, where there are often many small bitmaps encoded separately, this savings can be significant.

NOTE 2 – A decoder can take a number of approaches to deal with EOFB in the cases where it is optional. These approaches take advantage of the known byte count and the fact that the EOFB, if present, is counted in this byte count. Two possible approaches are:

- Invoke the MMR decoding procedure, and always check for EOFB after the bitmap has been decoded. However, allow the MMR decoding procedure to examine no more bytes than are known to be part of the MMR-compressed data block. If the MMR decoding procedure runs out of data while checking for EOFB, this is not an error, but a normal condition indicating that EOFB was not present.
- Invoke the MMR decoding procedure, and never check for EOFB after the bitmap has been decoded, in the cases where EOFB is optional. If the MMR decoding procedure consumes fewer bytes than are known to part of the MMR-compressed data block, this is not an error, but a normal condition indicating that EOFB was present. Skip over such unconsumed bytes.
 - The extension codes of T.6, including uncompressed mode, must not be present in the MMR-encoded data.

NOTE 3 – MMR provides less compression than image bitmap compression based on arithmetic coding. Image bitmap decoding using MMR is faster than image bitmap decoding based on arithmetic coding.

6.2.7 Colour extension of generic region segment

If **COLEXTFLAG** is **1**, after the decoding procedure described in 6.2.5 or 6.2.6, read **GBFGCOLID** which is a 4-byte integer indicating the colour palette ID, and set the colour specified by **GBCOLS**[**GBFGCOLID**] to the segment's foreground colour.

The background colour of the segment of which **COLEXTFLAG** is 1 is regarded as transparent.

NOTE – GBFGCOLID (the foreground colour information) has a fixed 4-byte field and is put at the end of the region segment only when **COLEXTFLAG** is 1. The decoder can find the address of GBFGCOLID by subtracting 4 from the segment data length recorded in the segment header.

6.3 Generic Refinement Region Decoding Procedure

6.3.1 General description

This decoding procedure is used to decode a rectangular array of **0** or **1** values, which are coded one pixel at a time. There is a reference bitmap known to the decoding procedure, and this is used as part of the decoding process. The reference bitmap is intended to resemble the bitmap being decoded, and this similarity is used to increase compression. Each pixel is decoded using a context comprising pixels drawn from the reference bitmap as well as previously-decoded pixels from the bitmap being decoded.

6.3.2 Input parameters

The parameters to this decoding procedure are shown in Table 6.

Table 6 – Parameters for the generic refinement region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
GRW	Integer	32	N	The width of the region.
GRH	Integer	32	N	The height of the region.
GRTEMPLATE	Integer	1	N	The template identifier.
GRREFERENCE	Bitmap			The reference bitmap.
GRREFERENCEDX	Integer	32	Y	The X offset of the reference bitmap with respect to the bitmap being decoded.
GRREFERENCEDY	Integer	32	Y	The Y offset of the reference bitmap with respect to the bitmap being decoded.
TPGRON	Integer	1	N	Whether typical prediction for generic refinement is used.
GRATX₁	Integer	8	Y	The X location of the adaptive template pixel RA ₁ . ^{a)}
GRATY₁	Integer	8	Y	The Y location of the adaptive template pixel RA ₁ . ^{a)}
GRATX₂	Integer	8	Y	The X location of the adaptive template pixel RA ₂ . ^{a)}
GRATY₂	Integer	8	Y	The Y location of the adaptive template pixel RA ₂ . ^{a)}

^{a)} Unused if **GRTEMPLATE** ≠ 0.

6.3.3 Return value

The variable whose value is the result of this decoding procedure is shown in Table 7.

Table 7 – Return value from the generic refinement region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
GRREG	Bitmap			The decoded region bitmap.

6.3.4 Variables used in decoding

The variables used by this decoding procedure are shown in Table 8.

Table 8 – Variables used in the generic refinement region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
CONTEXT	Integer	13	N	The values of the pixels in the template.
LTP	Integer	1	N	Whether the current image line is decoded explicitly. ^{a)}
SLTP	Integer	1	N	Whether the current line's LTP value is different from the previous line's LTP value. ^{a)}
TPGRPIX	Integer	1	N	Whether the current pixel is to be decoded implicitly using a TPGR prediction. ^{a)}
TPGRVAL	Integer	1	N	Value of the TPGR-predicted current pixel. ^{a)}
^{a)} Unused if TPGRON = 0.				

6.3.5 Decoding using a template and arithmetic coding

6.3.5.1 General description

The generic refinement region decoding procedure is based on arithmetic coding with a template to determine the coding state. The remainder of 6.3.5 describes this form of decoding.

6.3.5.2 Coding order and edge conventions

The coding algorithm iterates through the refined bitmap being decoded, along with a reference bitmap, in raster scan order. That is, it iterates by rows from top to bottom, and within each row from left to right. The processing for a current target pixel will reference some pixels in fixed spatial relationship to the target pixel. Some of these pixels are drawn from the reference version of the bitmap, and some of these pixels are drawn from the already-coded pixels of the refined bitmap.

Near the edges of the bitmap, these neighbour references might not lie in the actual bitmap. The rule to satisfy out-of-bounds references shall be:

- All pixels lying outside the bounds of the actual bitmap or the reference bitmap have the value 0.

6.3.5.3 Fixed templates and adaptive templates

A template defines a neighbourhood around a pixel to be coded. The values of the pixels in this neighbourhood define a context. Each context has its own adaptive probability estimate used by the arithmetic coder (see Annex E). Although a template is a geometric pattern of pixels, the pixels in a template are said to take on values when the template is aligned with a particular part of the image.

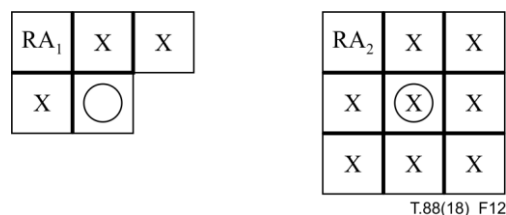


Figure 12 – 13-pixel refinement template showing the AT pixels at their nominal locations

Figure 12 shows the template which shall be used when GRTEMPLATE is 0. Figure 13 shows the template which shall be used when GRTEMPLATE is 1. In each of these figures, the left-hand group indicates the pixels from the already-coded pixels of the refined bitmap that are in the template, and the right-hand group indicates the pixels from the reference version of the template that are in the template. Each group in each figure includes a pixel denoted by a circle;

these pixels all correspond to the pixel to be coded. The pixels marked with an 'X' correspond to ordinary pixels in the template. The pixels denoted RA1-RA2 are special pixels in the template. They are denoted "adaptive" or AT pixels. These pixels are special in that their locations are allowed to change during the process of encoding the image. See 6.3.5.4 for a description of AT pixels. The legends RA1-RA2 indicate the nominal locations of AT pixels 1 to 2.

The AT pixel RA1 can be located anywhere in the field shown in Figure 7, not including the current pixel. The AT pixel RA2 can be located anywhere in the range (−128, −128) to (127, 127) in the reference bitmap.

The pixels in the left hand group of each template shall be aligned with the already-decoded pixels of the bitmap being decoded, with the pixel denoted by a circle lying on the pixel to be decoded. Let (X, Y) be the location of this pixel. The pixels of the right hand group of each template shall be aligned with the reference bitmap **GRREFERENCE**, with the pixel denoted by a circle placed at the location $(X - \mathbf{GRREFERENCEDX}, Y - \mathbf{GRREFERENCEDY})$. The values of the pixels in the template shall be combined to form a context. Each pixel in the template (including the adaptive pixels) shall correspond to a specific bit in the context, although the pixels in the template may be assigned to bits in the context in any order. Because there are up to 13 pixels in the template, contexts can take on up to 8192 different values. This context shall be used to identify which adaptive probability estimate shall be used by the arithmetic coder for encoding the pixel to be coded (see Annex E).

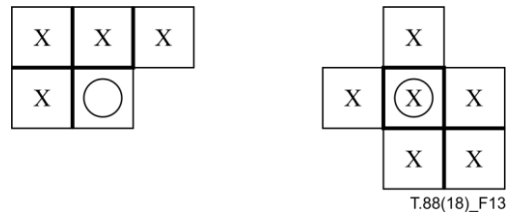


Figure 13 – 10-pixel refinement template

6.3.5.4 Adaptive template pixels

In coding the image, the template shall be allowed to change in the restricted way described in this clause.

The pixels that are allowed to change shall be called AT pixels. Their nominal locations are indicated by 'RA₁' and 'RA₂' in Figure 12. Note that only one template has AT pixels.

6.3.5.5 Typical prediction for generic refinement (TPGR)

Typical prediction for generic refinement can be enabled or disabled with the **TPGRON** parameter. If typical prediction for generic refinement is enabled (**TPGRON** is **1**) then before the first pixel of each row is decoded, a value indicating whether a row is typical shall be decoded. If the row is not typical, each pixel of the row needs to be explicitly decoded. If the row is typical, all typically-predictable pixels can be implicitly decoded using their predicted value, with the remainder of the pixels still being explicitly decoded. For a pixel to be typically-predictable it must meet the criteria defined in 6.3.5.6, step 3d).

NOTE – The value decoded before the first pixel of each row is not used in any pixel's template.

6.3.5.6 Decoding the refinement bitmap

The decoding of the bitmap proceeds as follows:

- 1) Set **LTP** = **0**.
- 2) Create a bitmap **GRREG** of width **GRW** and height **GRH** pixels.
- 3) Decode each row as follows:
 - a) If all **GRH** rows have been decoded, then the decoding is complete; proceed to step 4).
 - b) If **TPGRON** is **1**, then decode a bit using the arithmetic entropy coder, where the context used to decode this bit varies depending on the template in use:
 - If **GRTEMPLATE** is **0**, use the context shown in Figure 14.
 - If **GRTEMPLATE** is **1**, use the context shown in Figure 15.

Let **SLTP** be the value of this decoded bit. Set:

$$\mathbf{LTP} = \mathbf{LTP} \text{ XOR } \mathbf{SLTP}$$

- c) If **LTP** = **0** then, from left to right, explicitly decode all pixels of the current row of **GRREG**. The procedure for each pixel is as follows:
 - i) Place the template given by parameters **GRTEMPLATE** (and **GRATX₁**, **GRATY₁**, **GRATX₂** and **GRATY₂** if **GRTEMPLATE** is **0**) so that the current pixel is aligned with the location denoted by a circle in the figure describing the appearance of the template with identifier **GRTEMPLATE**.
 - ii) Form an integer **CONTEXT** by gathering the values of the image pixels overlaid by the template (including AT pixels) at its current location. The order of this gathering is not standardized, but must be consistent and independent of the location of the AT pixels.

- iii) Decode the current pixel by invoking the arithmetic entropy decoding procedure, with CX set to the value formed by concatenating the label "GR" and the 10-13 pixel values gathered in CONTEXT. The result of this invocation is the value of the current pixel.

EXAMPLE – If **GRTEMPLATE** is **1**, the image pixels overlaid by the template are as shown in Figure 15, and the pixels are gathered in reading order (in rows from top to bottom, and within each row from left to right, with the pixels in GRREG considered before the pixels in GRREFERENCE), then CX is set to "GR0000001000".

- d) If **LTP** = **1** then, from left to right, implicitly decode certain pixels of the current row of GRREG, and explicitly decode the rest. The procedure for each pixel is as follows:
- Set **TPGRPIX** equal to **1** if:
 - TPGRON** is **1** AND;
 - a 3×3 pixel array in the reference bitmap (Figure 16), centred at the location corresponding to the current pixel, contains pixels all of the same value.

When **TPGRPIX** is set to **1**, set **TPGRVAL** equal to the current pixel predicted value, which is the common value of the nine adjacent pixels in the 3×3 array.
 - If **TPGRPIX** is **1** then implicitly decode the current pixel by setting it equal to its predicted value (**TPGRVAL**).
 - Otherwise, explicitly decode the current pixel using the methodology of steps 3 c) i) through 3 c) iii) above.
- 4) After all the rows have been decoded, the current contents of the bitmap GRREG are the results that shall be obtained by every decoder, whether it performs this exact sequence of steps or not.

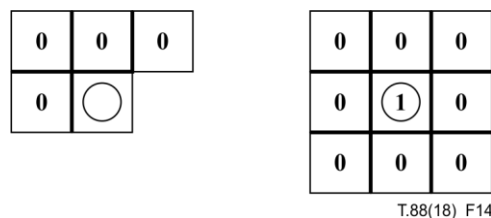


Figure 14 – Reused context for coding the SLTP value when **GRTEMPLATE** is **0**

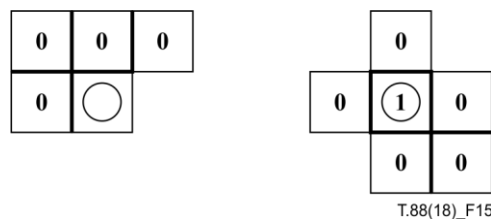


Figure 15 – Reused context for coding the SLTP value when **GRTEMPLATE** is **1**

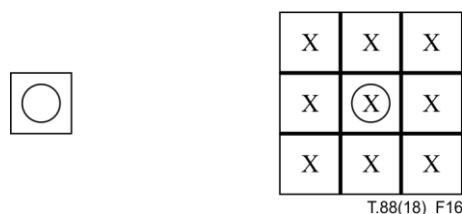


Figure 16 – **TPGR** template

6.4 Text Region Decoding Procedure

6.4.1 General description

This decoding procedure is used to decode a bitmap by decoding a number of symbol instances. A symbol instance contains a location and a symbol ID, and possibly a refinement bitmap. These symbol instances are combined to form the decoded bitmap.

NOTE – This decoding procedure will normally be used to decode the text part of a page. The symbols are normally single text characters from some font or alphabet.

6.4.2 Input parameters

The parameters to this decoding procedure are shown in Table 9.

NOTE – The values of some of these parameters in a typical situation, where a bitmap containing text characters in standard English reading order is being decoded, and 1 is the foreground pixel value, are:

- **SBDEFPIXEL** is 0
- **SBCOMBOP** is OR
- **TRANPOSED** is 0
- **REFCORNER** is BOTTOMLEFT

Table 9 – Parameters for the text region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
SBHUFF	Integer	1	N	Whether Huffman coding is used.
SBREFINE	Integer	1	N	Whether refinement coding is used.
SBW	Integer	32	N	The width of the region.
SBH	Integer	32	N	The height of the region.
SBNUMINSTANCES	Integer	32	N	The number of symbol instances in this region.
SBSTRIPS	Integer	4	N	The size of the symbol instance strips. May take on the values 1, 2, 4 or 8.
SBNUMSYMS	Integer	32	N	The number of symbols that may be used in this region.
SBSYMCODES	Array of Huffman codes			An array containing the codes for the symbols used in this region. Contains SBNUMSYMS codes. ^{a)}
SBSYMCODELEN	Integer	6	N	The length of the symbol codes used in IAID. ^{d)}
SBSYMS	Array of symbols			An array containing the symbols used in this text region. Contains SBNUMSYMS symbols.
SBDEFPIXEL	Integer	1	N	The default pixel for this bitmap.
SBCOMBOP	Operator			The combination operator for this text region. May take on the values OR, AND, XOR, XNOR and REPLACE. ^{e)}
TRANPOSED	Integer	1	N	Whether the strips run vertically.
REFCORNER	Corner			The reference corner of each symbol instance bitmap. May take on the values TOPLEFT, TOPRIGHT, BOTTOMLEFT and BOTTOMRIGHT.
SBDSOFFSET	Integer	5	Y	An offset for all the delta S values.
SBHUFFFS	Huffman table			The Huffman table used to decode the S coordinate of the first symbol instance in each strip. ^{a)}
SBHUFFFDS	Huffman table			The Huffman table used to decode the S coordinate of subsequent symbol instances in each strip. ^{a)}
SBHUFFDT	Huffman table			The Huffman table used to decode the difference in T coordinates between non-empty strips. ^{a)}
SBHUFFRDW	Huffman table			The Huffman table used to decode the difference between a symbol's width and the width of a refinement coded bitmap. ^{b)}
SBHUFFRDH	Huffman table			The Huffman table used to decode the difference between a symbol's height and the height of a refinement coded bitmap. ^{b)}
SBHUFFRDY	Huffman table			The Huffman table used to decode the difference between a symbol instance's X coordinate and the X coordinate of a refinement coded bitmap. ^{b)}

Table 9 – Parameters for the text region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
SBHUFFRDY	Huffman table			The Huffman table used to decode the difference between a symbol instance's Y coordinate and the Y coordinate of a refinement coded bitmap. ^{b)}
SBHUFFRSIZE	Huffman table			The Huffman table used to decode the size of a symbol instance's refinement bitmap data. ^{b)}
SBRTTEMPLATE	Integer	1	N	Template identifier for refinement coding of symbol instance bitmaps. ^{c)}
SBRTX₁	Integer	8	Y	The X location of the adaptive template pixel RA ₁ . ^{c)}
SBRTY₁	Integer	8	Y	The Y location of the adaptive template pixel RA ₁ . ^{c)}
SBRTX₂	Integer	8	Y	The X location of the adaptive template pixel RA ₂ . ^{c)}
SBRTY₂	Integer	8	Y	The Y location of the adaptive template pixel RA ₂ . ^{c)}
COLEXTFLAG	Integer	1	N	A parameter indicating whether the generic region segment is extended to represent coloured bitmap.
SBCOLS	Array of colours			An array containing the colours used in this text region. ^{f)}

a) Unused if **SBHUFF** = **0**.

b) Unused if **SBHUFF** = **0** or **SBREFINE** = **0**.

c) Unused if **SBREFINE** = **0**.

d) Unused if **SBHUFF** = **1**.

e) REPLACE operator is used if and only if **COLEXTFLAG** = **1**.

f) Unused if **COLEXTFLAG** = **0**.

6.4.3 Return value

The variable whose value is the result of this decoding procedure is shown in Table 10.

Table 10 – Return value from the text region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
SBREG	Bitmap			The decoded region bitmap.

6.4.4 Variables used in decoding

The variables used by this decoding procedure are shown in Table 11.

Table 11 – Variables used in the text region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
STRIPT	Integer	32	Y	The numerically smallest T coordinate in the current strip.
FIRSTS	Integer	32	Y	The first S coordinate of the current strip.
NINSTANCES	Integer	32	N	A symbol instance counter.
DT	Integer	32	Y	The number of empty strips between two non-empty strips.
DFS	Integer	32	Y	The difference in S coordinates between the first symbol instances of two strips.
CURS	Integer	32	Y	The current S coordinate.
CURT	Integer	3	N	The current symbol instance's T coordinate relative to the current strip.
S_I	Integer	32	Y	A symbol instance's S coordinate.
T_I	Integer	32	Y	A symbol instance's T coordinate.
ID_I	Integer	32	N	A symbol instance's symbol ID.
IB_I	Bitmap			A symbol instance's symbol bitmap.
W_I	Integer	32	N	The width of a symbol instance's symbol bitmap.
H_I	Integer	32	N	The height of a symbol instance's symbol bitmap.

Table 11 – Variables used in the text region decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
IDS	Integer	32	Y	The difference in S coordinates between two symbol instances within a strip.
R_I	Integer	1	N	Whether a symbol instance's symbol bitmap is coded using refinement.
RDW_I	Integer	32	Y	The delta width of a symbol instance's refinement bitmap. ^{a)}
RDH_I	Integer	32	Y	The delta height of a symbol instance's refinement bitmap. ^{a)}
RDX_I	Integer	32	Y	The delta X of a symbol instance's refinement bitmap. ^{a)}
RDY_I	Integer	32	Y	The delta Y of a symbol instance's refinement bitmap. ^{a)}
IBO_I	Bitmap			A symbol instance's original symbol bitmap. ^{a)}
WO_I	Integer	32	N	The width of IBO_I . ^{a)}
HO_I	Integer	32	N	The height of IBO_I . ^{a)}
SBCOLSECTSIZE	Integer	32	N	The colour section length. ^{b)}
SBFGCOLID	Integer	32	N	An array of colour palette ID, indicating the colour of each symbol instance. ^{b)}
^{a)} Unused if SBREFINE = 0. ^{b)} Unused if COLEXTFLAG = 0.				

6.4.5 Decoding the text region

A symbol-coded bitmap is represented by a set of symbol instances. Each symbol instance encodes a location, a symbol ID, and possibly refinement information. The location of each symbol instance comprises an S coordinate and a T coordinate. If **TRANSPPOSED** is **0**, then the S coordinate axis corresponds to the X axis of the bitmap, and the T axis corresponds to the Y axis of the bitmap. If **TRANSPPOSED** is **1**, then the S coordinate axis corresponds to the Y axis of the bitmap, and the T axis corresponds to the X axis of the bitmap.

NOTE 1 – Transposing the coordinate axes allows efficient coding of text running vertically. The reference corner is variable because the most efficient coding is usually obtained when the reference corner of each symbol instance lies on a text baseline, and the text baselines may run in any direction.

In order to improve compression, symbol instances are grouped into strips according to their T_I values. This is done according to the value of **SBSTRIPS**. Symbol instances having T_I values between 0 and **SBSTRIPS** – 1 are grouped into one strip, symbol instances having T_I values between **SBSTRIPS** and $2 \times \text{SBSTRIPS} - 1$ into the next, and so on. Within each strip, the symbol instances are coded in the order of increasing S coordinate.

NOTE 2 – Normally the strips occur in the order of strictly increasing T coordinates, and the symbol instances within each strip occur in the order of nondecreasing S coordinates. However, it is possible for negative delta S or delta T values to occur during the decoding, meaning that the strips and symbol instances might occur in any order.

The overall structure of the data to be decoded in order to reconstruct the text region is shown in Figure 17. The format of each strip is as shown in Figure 19. When **SBREFINE** is **0**, the format of each symbol instance is as shown in Figure 20. When **SBREFINE** is **1**, the format of each symbol instance is as shown in Figure 21.

NOTE 3 – There may be some symbol instances whose reference corner lies off the top of the region. If these are to be coded, there must be some way to have a strip that also lies above the top of the region. The initial value of STRIPT is the coordinate with respect to which the first strip is located.

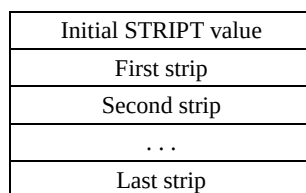


Figure 17 – Coded structure of a text region

Initial STRIPT values
First strip
Second strip
...
Last strip
Colour section

Figure 18 – Coded structure of a text region extended for colour text

Delta T
First symbol instance
Second symbol instance
...
Last symbol instance
OOB

Figure 19 – Structure of a strip

Symbol instance S coordinate
Symbol instance T coordinate
Symbol instance symbol ID

Figure 20 – Structure of a symbol instance when SBREFINE is 0

Symbol instance S coordinate
Symbol instance T coordinate
Symbol instance symbol ID
Symbol instance refinement information

Figure 21 – Structure of a symbol instance when SBREFINE is 1

Colour palette IDs
Colour section length

Figure 22 – Coded structure of colour section

NOTE 4 – If **COLEXTFLAG** is 1 then a colour section, which specifies the colours of all symbol instances, is put at the end of the region as shown in Figure 18. The colour section consists of two parts as shown in Figure 22: colour palette IDs and colour section length. The colour palette IDs, which is encoded in the data structure based on Recommendation ITU-T T.45, contains the information that specifies the colour of every symbol instances in the segment. The colour section length is a 4-byte field that contains the length of the colour section, in bytes.

NOTE 5 – The background colour of the segment of which **COLEXTFLAG** is 1 is regarded as transparent.

The result of decoding a text region shall be the bitmap that is produced by the following steps:

- 1) Fill a bitmap SBREG, of the size given by **SBW** and **SBH**, with the **SBDEFPIXEL** value.
- 2) Decode the initial STRIPT value as described in 6.4.6. Negate the decoded value and assign this negated value to the variable STRIPT. Assign the value 0 to FIRSTS. Assign the value 0 to NINSTANCES.
- 3) If **COLEXTFLAG** is 1, decode the colour section as described in 6.4.12.
- 4) Decode each strip as follows:
 - a) If NINSTANCES is equal to **SBNUMINSTANCES** then there are no more strips to decode, and the process of decoding the text region is complete; proceed to step 4).
 - b) Decode the strip's delta T value as described in 6.4.6. Let DT be the decoded value. Set:

$$\text{STRIPT} = \text{STRIPT} + \text{DT}$$

- c) Decode each symbol instance in the strip as follows:
- i) If the current symbol instance is the first symbol instance in the strip, then decode the first symbol instance's S coordinate as described in 6.4.7. Let DFS be the decoded value. Set:

$$\begin{aligned}\text{FIRSTS} &= \text{FIRSTS} + \text{DFS} \\ \text{CURS} &= \text{FIRSTS}\end{aligned}$$

- ii) Otherwise, if the current symbol instance is not the first symbol instance in the strip, decode the symbol instance's S coordinate as described in 6.4.8. If the result of this decoding is OOB then the last symbol instance of the strip has been decoded; proceed to step 3 d). Otherwise, let IDS be the decoded value. Set:

$$\text{CURS} = \text{CURS} + \text{IDS} + \text{SBDSOFFSET}$$

NOTE 6 – The intended use of **SBDSOFFSET** is to make the most common value decoded in 6.4.8 zero. The shortest code in all of the default tables used in 6.4.8 is for the value zero.

- iii) Decode the symbol instance's T coordinate as described in 6.4.9. Let CURT be the decoded value. Set:

$$T_I = \text{STRIPT} + \text{CURT}$$

- iv) Decode the symbol instance's symbol ID as described in 6.4.10. Let ID_I be the decoded value.
- v) Determine the symbol instance's bitmap IB_I as described in 6.4.11. The width and height of this bitmap shall be denoted as W_I and H_I respectively.
- vi) Update CURS as follows:

- If **TRANPOSED** is **0**, and **REFCORNER** is TOPRIGHT or BOTTOMRIGHT, set:

$$\text{CURS} = \text{CURS} + W_I - 1$$

- If **TRANPOSED** is **1**, and **REFCORNER** is BOTTOMLEFT or BOTTOMRIGHT, set:

$$\text{CURS} = \text{CURS} + H_I - 1$$

- Otherwise, do not change CURS in this step.

- vii) Set:

$$S_I = \text{CURS}$$

- viii) Determine the location of the symbol instance bitmap with respect to SBREG as follows:

- If **TRANPOSED** is **0**, then:
 - If **REFCORNER** is TOPLEFT then the top left pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[S_I, T_I]$.
 - If **REFCORNER** is TOPRIGHT then the top right pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[S_I, T_I]$.
 - If **REFCORNER** is BOTTOMLEFT then the bottom left pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[S_I, T_I]$.
 - If **REFCORNER** is BOTTOMRIGHT then the bottom right pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[S_I, T_I]$.
- If **TRANPOSED** is **1**, then:
 - If **REFCORNER** is TOPLEFT then the top left pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[T_I, S_I]$.
 - If **REFCORNER** is TOPRIGHT then the top right pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[T_I, S_I]$.
 - If **REFCORNER** is BOTTOMLEFT then the bottom left pixel of the symbol instance bitmap IB_I shall be placed at $\text{SBREG}[T_I, S_I]$.

- If **REFCORNER** is BOTTOMRIGHT then the bottom right pixel of the symbol instance bitmap IB_I shall be placed at $SBREG[T_I, S_I]$.

If any part of IB_I , when placed at this location, lies outside the bounds of SBREG, then ignore this part of IB_I in step 3 c) ix).

- ix) If **COLEXTFLAG** is 1, set the colour specified by **SBCOLS**[SBFGCOLID[NINSTANCES]] to the foreground colour of the symbol instance bitmap IB_I .
- x) Draw IB_I into SBREG. Combine each pixel of IB_I with the current value of the corresponding pixel in SBREG, using the combination operator specified by **SBCOMBOP**. Write the results of each combination into that pixel in SBREG.
- xi) Update CURS as follows:
 - If **TRANPOSED** is 0, and **REFCORNER** is TOPLEFT or BOTTOMLEFT, set:

$$CURS = CURS + W_I - 1$$

- If **TRANPOSED** is 1, and **REFCORNER** is TOPLEFT or TOPRIGHT, set:

$$CURS = CURS + H_I - 1$$

- Otherwise, do not change CURS in this step.

NOTE 7 – The CURS update rules are designed to allow the gap between adjacent symbol instances to be encoded, rather than the distance between their reference corners; this takes out one source of variation (the symbol instance bitmap width or height), and allows better compression.

- xii) Set:

$$NINSTANCES = NINSTANCES + 1$$

- d) When the strip has been completely decoded, decode the next strip.
- 5) After all the strips have been decoded, the current contents of SBREG are the results that shall be obtained by every decoder, whether it performs this exact sequence of steps or not.

NOTE 8 – The colour palette (**SBCOLS**) is created by concatenating the default colour set (CPDEFCOLS) and the additional colours (CPEXCOLS) defined in the colour palette segment referred to by this segment.

6.4.6 Strip delta T

If **SBHUFF** is 1, decode a value using the Huffman table specified by **SBHUFFDT** and multiply the resulting value by **SBSTRIPS**.

If **SBHUFF** is 0, decode a value using the IADT integer arithmetic decoding procedure (see Annex A) and multiply the resulting value by **SBSTRIPS**.

6.4.7 First symbol instance S coordinate

NOTE – The symbol instance S coordinate value for the first symbol instance of each strip is coded differently from the subsequent symbol instances in each strip. This takes advantage of the beginnings of lines being aligned.

If **SBHUFF** is 1, decode a value using the Huffman table specified by **SBHUFFFS**.

If **SBHUFF** is 0, decode a value using the IAFS integer arithmetic decoding procedure (see Annex A).

6.4.8 Subsequent symbol instance S coordinate

If **SBHUFF** is 1, decode a value using the Huffman table specified by **SBHUFFDS**.

If **SBHUFF** is 0, decode a value using the IADS integer arithmetic decoding procedure (see Annex A).

In either case it is possible that the result of this decoding is the out-of-band value OOB.

6.4.9 Symbol instance T coordinate

If **SBSTRIPS** = 1, then the value decoded is always zero. Otherwise:

- If **SBHUFF** is 1, decode a value by reading $\lceil \log_2 \text{SBSTRIPS} \rceil$ bits directly from the bitstream.
- If **SBHUFF** is 0, decode a value using the IAIT integer arithmetic decoding procedure (see Annex A).

NOTE – If **SBSTRIPS** = 1, then no bits are consumed, and the IAIT integer arithmetic decoding procedure is never invoked.

6.4.10 Symbol instance symbol ID

If **SBHUFF** is **1**, decode a value by reading one bit at a time until the resulting bit string is equal to one of the entries in **SBSYMCODES**. The resulting value, which is ID_I , is the index of the entry in **SBSYMCODES** that is read.

If **SBHUFF** is **0**, decode a value using the IAID integer arithmetic decoding procedure (see Annex A). Set ID_I to the resulting value.

6.4.11 Symbol instance bitmap

In some cases, the symbol instance bitmap IB_I is simply the bitmap of the symbol identified by ID_I . In other cases, however, the symbol instance bitmap is that bitmap modified by additional refinement information. The bit indicating which of the options is true for a symbol instance is called R_I .

If **SBREFINE** is **0**, then set R_I to **0**.

If **SBREFINE** is **1**, then decode R_I as follows:

- If **SBHUFF** is **1**, then read one bit and set R_I to the value of that bit.
- If **SBHUFF** is **0**, then decode one bit using the IARI integer arithmetic decoding procedure and set R_I to the value of that bit.

If R_I is **0** then set the symbol instance bitmap IB_I to **SBSYMS**[ID_I].

If R_I is **1** then determine the symbol instance bitmap as follows:

- 1) Decode the symbol instance refinement delta width as described in 6.4.11.1. Let RDW_I be the value decoded.
- 2) Decode the symbol instance refinement delta height as described in 6.4.11.2. Let RDH_I be the value decoded.
- 3) Decode the symbol instance refinement X offset as described in 6.4.11.3. Let RDX_I be the value decoded.
- 4) Decode the symbol instance refinement Y offset as described in 6.4.11.4. Let RDY_I be the value decoded.
- 5) If **SBHUFF** is **1**, then:
 - a) Decode the symbol instance refinement bitmap data size as described in 6.4.11.5.
 - b) Skip over any bits remaining in the last byte read.
- 6) Let IBO_I be **SBSYMS**[ID_I]. Let WO_I be the width of IBO_I and HO_I be the height of IBO_I . The symbol instance bitmap IB_I is the result of applying the generic refinement region decoding procedure described in 6.3. Set the parameters to this decoding procedure as shown in Table 12.
- 7) If **SBHUFF** is **1**, then skip over any bits remaining in the last byte read. The total number of bytes processed by the generic refinement bitmap decoding procedure must be equal to the value read in step 5 a).

6.4.11.1 Symbol instance refinement delta width

This field, and the following fields, indicate the size, location and contents of the refined symbol bitmap, as the size might not be the same as the size of the bitmap of the symbol whose ID is given in this symbol instance; also, the change in the size of the bitmap might extend to the left and top, not just to the right and bottom, so we need to supply an offset as well as a size. Note that the offsets are given in terms of X and Y, not S and T.

If **SBHUFF** is **1**, decode a value using the Huffman table specified by **SBHUFFRDW**.

If **SBHUFF** is **0**, decode a value using the IARDW integer arithmetic decoding procedure (see Annex A).

Table 12 – Parameters used to decode a symbol instance's bitmap using refinement

Name	Value
GRW	$WO_I + RDW_I$
GRH	$HO_I + RDH_I$
GRTEMPLATE	SBRTEMPLATE
GRREFERENCE	IBO_I
GRREFERENCEDX	$\lfloor RDW_I/2 \rfloor + RDX_I$
GRREFERENCEDY	$\lfloor RDH_I/2 \rfloor + RDY_I$
TPGRON	0
GRATX ₁	SBRATX ₁
GRATY ₁	SBRATY ₁
GRATX ₂	SBRATX ₂
GRATY ₂	SBRATY ₂

6.4.11.2 Symbol instance refinement delta height

If **SBHUFF** is **1**, decode a value using the Huffman table specified by **SBHUFFRDH**.

If **SBHUFF** is **0**, decode a value using the IARDH integer arithmetic decoding procedure (see Annex A).

6.4.11.3 Symbol instance refinement X offset

If **SBHUFF** is **1**, decode a value using the Huffman table specified by **SBHUFFRDX**.

If **SBHUFF** is **0**, decode a value using the IARDX integer arithmetic decoding procedure (see Annex A).

6.4.11.4 Symbol instance refinement Y offset

If **SBHUFF** is **1**, decode a value using the Huffman table specified by **SBHUFFRDY**.

If **SBHUFF** is **0**, decode a value using the IARDY integer arithmetic decoding procedure (see Annex A).

6.4.11.5 Symbol instance refinement bitmap data size

Decode a value using the Huffman table specified by **SBHUFFRSIZE**.

6.4.12 Colour section

If **COLEXTFLAG** is **1**, the colour information recorded in the colour section needs to be decoded as shown in this clause.

6.4.12.1 Colour section length

The decoding procedure of the colour section begins with achieving its top address by subtracting the **SBCOLSECTSIZE** from the segment data length defined in the segment header. **SBCOLSECTSIZE** is a 4-byte field at the end of the text region segment data part. The decoder can find the end of the data part by reading the segment data length in the segment header.

6.4.12.2 Colour palette IDs

The field of the colour palette IDs is decoded in accordance with Recommendation ITU-T T.45 and the result is stored in **SBFGCOLID**. The number of elements in **SBFGCOLID** is equal to the number of symbol instances in the text region segment (**SBNUMINSTANCES**).

NOTE – If the codestream has the coloured text region segment, the ITU-T T.45 for colour palette IDs in the colour section is restricted as follows:

- The number of colour components is 1 (**NCOMP** = 1 in the header of ITU-T T.45 codestream).
- The length of each colour component's value is 1-byte (**COMPLEN** = 1 in the header of ITU-T T.45 codestream).
- The number of colour values is equal to **SBNUMINSTANCES** (**NVALS** = **SBNUMINSTANCES** in the header of ITU-T T.45 codestream).

6.5 Symbol Dictionary Decoding Procedure

6.5.1 General description

This decoding procedure is used to decode a set of symbols; these symbols can then be used by text region decoding procedures, or in some cases by other symbol dictionary decoding procedures.

6.5.2 Input parameters

The parameters to this decoding procedure are shown in Table 13.

The **SDREFAGG** parameter determines how the symbols in this symbol dictionary are coded. If **SDREFAGG** is **0** then each symbol bitmap is coded via direct bitmap coding. If **SDREFAGG** is **1** then each symbol bitmap is coded by refining or aggregating previously-defined symbol bitmaps. These previously-defined symbol bitmaps may be drawn from other dictionaries and provided as input to this decoding procedure in **SDINSYMS**, or may be defined in the current dictionary.

6.5.3 Return value

The variable whose value is the result of this decoding procedure is shown in Table 14.

6.5.4 Variables used in decoding

The variables used by this decoding procedure are shown in Table 15.

Table 13 – Parameters for the symbol dictionary decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
SDHUFF	Integer	1	N	Whether Huffman coding is used.
SDREFAGG	Integer	1	N	Whether refinement and aggregate coding are used.
SDNUMINSYMS	Integer	32	N	The number of symbols that are used as input to this symbol dictionary decoding procedure.
SDINSYMS	Array of symbols			An array containing the symbols that are used as input to this symbol dictionary decoding procedure. Contains SDNUMINSYMS symbols.
SDNUMNEWSYMS	Integer	32	N	The number of symbols to be defined in this symbol dictionary.
SDNUMEXSYMS	Integer	32	N	The number of symbols to be exported from this symbol dictionary.
SDHUFFDH	Huffman table			The Huffman table used to decode the difference in height between two height classes. ^{a)}
SDHUFFDW	Huffman table			The Huffman table used to decode the difference in width between two symbols. ^{a)}
SDHUFFBMSIZE	Huffman table			The Huffman table used to decode the size of a height class collective bitmap. ^{a)}
SDHUFFAGGINST	Huffman table			The Huffman table used to decode the number of symbol instances in an aggregation. ^{b)}
SDTEMPLATE	Integer	2	N	The template identifier used to decode symbol bitmaps. ^{c)}
SDATX₁	Integer	8	Y	The X location of the adaptive template pixel A ₁ . ^{c)}
SDATY₁	Integer	8	Y	The Y location of the adaptive template pixel A ₁ . ^{c)}
SDATX₂	Integer	8	Y	The X location of the adaptive template pixel A ₂ . ^{c)}
SDATY₂	Integer	8	Y	The Y location of the adaptive template pixel A ₂ . ^{c)}
SDATX₃	Integer	8	Y	The X location of the adaptive template pixel A ₃ . ^{c)}
SDATY₃	Integer	8	Y	The Y location of the adaptive template pixel A ₃ . ^{c)}
SDATX₄	Integer	8	Y	The X location of the adaptive template pixel A ₄ . ^{c)}
SDATY₄	Integer	8	Y	The Y location of the adaptive template pixel A ₄ . ^{c)}
SDRTEMPLATE	Integer	1	N	Template identifier for refinement coding of bitmaps. ^{d)}
SDRATX₁	Integer	8	Y	The X location of the adaptive template pixel RA ₁ . ^{d)}
SDRATY₁	Integer	8	Y	The Y location of the adaptive template pixel RA ₁ . ^{d)}
SDRATX₂	Integer	8	Y	The X location of the adaptive template pixel RA ₂ . ^{d)}
SDRATY₂	Integer	8	Y	The Y location of the adaptive template pixel RA ₂ . ^{d)}

Table 13 – Parameters for the symbol dictionary decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
a) Unused if SDHUFF = 0. b) Unused if SDHUFF = 0 or SDREFAGG = 0. c) Unused if SDHUFF = 1. d) Unused if SDREFAGG = 0.				

Table 14 – Return value from the symbol dictionary decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
SDEXSYMS	Array of symbols			The symbols exported by this symbol dictionary. Contains SDNUMEXSYMS symbols.

Table 15 – Variables used in the symbol dictionary decoding procedure

Name	Type	Size (bits)	Signed?	Description and restrictions
SDNEWSYMS	Array of symbols			The symbols defined in this symbol dictionary. Contains SDNUMNEWSYMS symbols.
SDNEWSYMWIDTHS	Array of integers			The widths of the symbols in SDNEWSYMS. Contains SDNUMNEWSYMS integers. Each integer is a 32-bit unsigned value.
HCHEIGHT	Integer	32	N	Height of the current height class.
NSYMSDECODED	Integer	32	N	How many symbols have been decoded so far.
HCDH	Integer	32	Y	The difference in height between two height classes.
SYMWIDTH	Integer	32	N	The width of the current symbol.
TOTWIDTH	Integer	32	N	The width of the current height class.
HCFIRSTSYM	Integer	32	N	The index of the first symbol in the current height class.
DW	Integer	32	Y	The difference in width between two symbols.
B_S	Bitmap			The current symbol's bitmap.
B_{HC}	Bitmap			The current height class collective bitmap.
I	Integer	32	N	An array index.
J	Integer	32	N	An array index.
REFAGGNINST	Integer	32	N	The number of symbol instances in an aggregation.
EXFLAGS	Array of integers			The export flags for this dictionary. Contains SDNUMINSYMS + SDNUMNEWSYMS values. Each value is one bit.
EXINDEX	Integer	32	N	An array index.
CUREXFLAG	Integer	1	N	The current export flag.
EXRUNLENGTH	Integer	32	N	The length of a run of identical export flag values.

6.5.5 Decoding the symbol dictionary

The internal structure of a symbol dictionary is shown in Figure 23. The symbols defined in the dictionary are ordered into height classes: a height class contains a number of symbols whose bitmaps are the same height.

NOTE 1 – In most cases, the height classes occur in the order of strictly increasing height, shortest through tallest. If **SDREFAGG** is 1, though, a symbol may be coded as a refinement of a larger symbol defined in the same dictionary. In this case, the height class for that base symbol must be decoded (and therefore must occur) before the shorter height class of the symbol that is coded by refining it. For this reason, height class delta heights (and symbol delta widths) may be zero or negative, as well as positive.

First height class
Second height class
...
Last height class
List of exported symbols

Figure 23 – The structure of a symbol dictionary

If **SDHUFF** is **1** and **SDREFAGG** is **0** then the format of a height class is as shown in Figure 24. Otherwise, the format of a height class is as shown in Figure 25. The fields mentioned in those figures are described fully below.

Height class delta height
Delta width for first symbol
Delta width for second symbol
...
OOB
Height class collective bitmap

Figure 24 – Height class coding when SDHUFF is 1 and SDREFAGG is 0

Height class delta height
Delta width for first symbol
Bitmap for first symbol
Delta width for second symbol
Bitmap for second symbol
...
OOB

Figure 25 – Height class coding when SDHUFF is 0 or SDREFAGG is 1

The result of decoding a symbol dictionary is an array **SDEXXSYMS** containing **SDNUMEXSYMS** bitmaps. This array shall be the array produced by the following steps:

- 1) Create an array **SDNEWSYMS** of bitmaps, having **SDNUMNEWSYMS** entries.
- 2) If **SDHUFF** is **1** and **SDREFAGG** is **0**, create an array **SDNEWSYMWIDTHS** of integers, having **SDNUMNEWSYMS** entries.
- 3) Set:

$$\begin{aligned}\text{HCHEIGHT} &= 0 \\ \text{NSYMSDECODED} &= 0\end{aligned}$$

- 4) Decode each height class as follows:
 - a) If **NSYMSDECODED** = **SDNUMNEWSYMS** then all the symbols in the dictionary have been decoded; proceed to step 5).
 - b) Decode the height class delta height as described in 6.5.6. Let **HCDH** be the decoded value. Set:

$$\begin{aligned}\text{HCHEIGHT} &= \text{HHEIGHT} + \text{HCDH} \\ \text{SYMWIDTH} &= 0 \\ \text{TOTWIDTH} &= 0 \\ \text{HCFIRSTSYM} &= \text{NSYMSDECODED}\end{aligned}$$
 - c) Decode each symbol within the height class as follows:
 - i) Decode the delta width for the symbol as described in 6.5.7. If the result of this decoding is OOB then all the symbols in this height class have been decoded; proceed to step 4 d). Otherwise let **DW** be the decoded value and set:

$$\begin{aligned}\text{SYMWIDTH} &= \text{SYMWIDTH} + \text{DW} \\ \text{TOTWIDTH} &= \text{TOTWIDTH} + \text{SYMWIDTH}\end{aligned}$$

- ii) If **SDHUFF** is **0** or **SDREFAGG** is **1**, then decode the symbol's bitmap as described in 6.5.8. Let B_S be the decoded bitmap (this bitmap has width SYMWIDTH and height HCHEIGHT). Set:

$$\text{SDNEWSYMS}[\text{NSYMSDECODED}] = B_S$$

- iii) If **SDHUFF** is **1** and **SDREFAGG** is **0**, then set:

$$\text{SDNEWSYMWIDTHS}[\text{NSYMSDECODED}] = \text{SYMWIDTH}$$

- iv) Set:

$$\text{NSYMSDECODED} = \text{NSYMSDECODED} + 1$$

- d) If **SDHUFF** is **1** and **SDREFAGG** is **0**, then decode the height class collective bitmap as described in 6.5.9. Let B_{HC} be the decoded bitmap. This bitmap has width TOTWIDTH and height HCHEIGHT. Break up the bitmap B_{HC} as follows to obtain the symbols $\text{SDNEWSYMS}[\text{HCFIRSTSYM}]$ through $\text{SDNEWSYMS}[\text{NSYMSDECODED} - 1]$.

B_{HC} contains the $\text{NSYMSDECODED} - \text{HCFIRSTSYM}$ symbols concatenated left-to-right, with no intervening gaps. For each I between HCFIRSTSYM and $\text{NSYMSDECODED} - 1$:

- the width of $\text{SDNEWSYMS}[I]$ is the value of $\text{SDNEWSYMWIDTHS}[I]$;
- the height of $\text{SDNEWSYMS}[I]$ is HCHEIGHT; and
- the bitmap $\text{SDNEWSYMS}[I]$ can be obtained by extracting the columns of B_{HC} from:

$$\sum_{J=\text{HCFIRSTSYM}}^{I-1} \text{SDNEWSYMWIDTHS}[J]$$

through

$$\left(\sum_{J=\text{HCFIRSTSYM}}^{I-1} \text{SDNEWSYMWIDTHS}[J] \right)^{-1}$$

EXAMPLE – Columns 0 through $\text{SDNEWSYMWIDTHS}[\text{HCFIRSTSYM}] - 1$ of B_{HC} contain the bitmap for $\text{SDNEWSYMS}[\text{HCFIRSTSYM}]$, the first symbol in the height class.

- 5) Determine which symbol bitmaps are exported from this symbol dictionary, as described in 6.5.10. These bitmaps can be drawn from the symbols that are used as input to the symbol dictionary decoding procedure as well as the new symbols produced by the decoding procedure.

NOTE 2 – Not all the new symbols need to be exported; this allows the dictionary to define a symbol, use it via refinement/aggregate coding to build other symbols, and not actually export the original symbol. Also, since input symbols can be exported, this dictionary can, in effect, copy symbols from other dictionaries.

6.5.6 Height class delta height

If **SDHUFF** is **1**, decode a value using the Huffman table specified by **SDHUFFDH**.

If **SDHUFF** is **0**, decode a value using the IADH integer arithmetic decoding procedure (see Annex A).

6.5.7 Delta width

If **SDHUFF** is **1**, decode a value using the Huffman table specified by **SDHUFFDW**.

If **SDHUFF** is **0**, decode a value using the IADW integer arithmetic decoding procedure (see Annex A).

In either case it is possible that the result of this decoding is the out-of-band value OOB.

6.5.8 Symbol bitmap

This field is only present if **SDHUFF** = **0** or **SDREFAGG** = **1**. This field takes one of two forms; **SDREFAGG** determines which form is used.