International Standard

ISO/IEC 16022

Third edition
2024-05

Information technology — Automatic identification and data capture techniques — Data Matrix bar code symbology specification

Technologies de l'information — Techniques automatiques d'identification et de capture des données — Spécification de symbologie de code à barres Data Matrix

Reference number
ISO/IEC 16022:2024(en)

Contents

Page

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 documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

This third edition cancels and replaces the second edition (ISO/IEC 16022:2006), which has been technically revised.

The main changes are as follows:

— the extended channel interpretations and rectangular formats have become a mandatory feature;

— the historic data matrix variant "ECC 000" to "ECC 140" has been removed;

— continuous grading according to ISO/IEC 15415 has been introduced to all quality measurements;

— transition ratio grading has been changed;

— new quality parameter “print growth” has been added;

— the reference decode algorithm has been revised;

— the interleaving blocks for 144 x 144 matrix size have been clarified.

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 and www.iec.ch/national-committees.

Introduction

Data Matrix is a two-dimensional matrix symbology which is made up of nominally square modules arranged within a perimeter finder pattern. Though primarily shown and described in this document as a dark symbol on light background, Data Matrix symbols can also be printed to appear as light on dark.

Manufacturers of bar code equipment and users of the technology need publicly available standard symbology specifications to which they can refer when developing equipment and application standards. The publication of standardised symbology specifications is designed to achieve this.

Information technology — Automatic identification and data capture techniques — Data Matrix bar code symbology specification

1 Scope

This document defines the requirements for the symbology known as Data Matrix. It specifies the Data Matrix symbology characteristics, data character encodation, symbol formats, dimensions and print quality requirements, error correction rules, decoding algorithm, and user-selectable application parameters.

It applies to all Data Matrix symbols produced by any printing or marking technology.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 19762, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary*

ISO/IEC 15415, *Information technology — Automatic identification and data capture techniques — Bar code symbol print quality test specification — Two-dimensional symbols*

ISO/IEC 646, *Information technology — ISO 7-bit coded character set for information interchange*

ISO/IEC 8859-1, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*

ISO/IEC 29158, *Information technology — Automatic identification and data capture techniques — Direct Part Mark (DPM) Quality Guideline*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
codeword
symbol character value
intermediate level of coding between source data and the graphical encodation in the symbol

3.2
module
single cell of element in a matrix symbology symbol used to encode one bit of the *codeword* (3.1)

[SOURCE: ISO/IEC 19762:2016, 04.02.06]

3.3
pattern randomising
procedure to convert an original bit pattern to another bit pattern by inverting selected bits

Note 1 to entry: The resulting bitstream is less likely to have repeating patterns.

4 Symbols

e	number of erasures
k	total number of error correction codewords
n	total number of data codewords
N	numerical base in an encodation scheme
p	number of codewords reserved for error detection
S	symbol character
t	number of errors
X	horizontal and vertical width of a module
ε	error correction codeword

5 Mathematical or logical notations

div	integer division operator
mod	integer remainder after division
XOR	exclusive-or logic function whose output is one only when its two inputs are not equivalent

6 Symbol description

6.1 Basic characteristics

Data Matrix is a two-dimensional matrix symbology.

The characteristics of Data Matrix are:

a) Encodable character set:

   1) values 0 – 127 in accordance with ISO/IEC 646 IRV, i.e. all 128 ASCII characters;

   2) values 128 - 255 in accordance with ISO/IEC 8859-1 (these are referred to as extended ASCII);

   3) additional characters can be encoded using the ECI capabilities.

b) Representation of data: A dark module is a binary one and a light module is a zero.

   This document specifies Data Matrix symbols in terms of dark modules marked on a light background. However, 6.2 provides that symbols can also be produced with the module's colours reversed. In such symbols, dark modules would be a binary zero, and light modules would be a binary one.

c) Symbol size in modules (not including quiet zone) ranging from 10 x 10 to 144 x 144 square and rectangular versions ranging from 8 x 18 to 16 x 48 (see Table 10).

   NOTE Additional rectangular symbol sizes are defined in ISO/IEC 21471 (see Reference [4]).

d) Data characters per symbol (for maximum symbol size):

   1) Alphanumeric data: up to 2 335 characters

   2) 8-bit byte data: 1 555 characters

   3) Numeric data: 3 116 digits.

e) Code type: Matrix

f) Orientation independence: Yes

g) Error detection and correction: Reed Solomon.

6.2 Summary of additional features

The following summarises additional features which are inherent or optional in Data Matrix:

a) Reflectance reversal: (inherent): Symbols are either dark on light or light on dark (see Figure 1). The specifications in this document are based on dark images on a light background, therefore references to dark or light modules should be taken as references to light or dark modules respectively in the case of symbols produced with reflectance reversal.

b) Extended Channel Interpretations: (ECI), (inherent): This mechanism enables characters from other character sets (e.g. Arabic, Cyrillic, Greek, Hebrew) and other data interpretations or industry-specific requirements to be represented.

c) Rectangular symbols: (inherent): Six symbol formats are specified in a rectangular form.

   NOTE Additional rectangular symbol formats are available by ISO/IEC 21471 (see Reference [4]).

d) Structured append: (optional): This allows files of data to be represented in up to 16 Data Matrix symbols. The original data can be correctly reconstructed regardless of the order in which the symbols are scanned. If the feature is not implemented, reader should not transmit data in case of a structured append symbol.

6.3 Symbol structure

6.3.1 General

Each Data Matrix symbol consists of data regions which contain nominally square modules set out in a regular array. In larger symbols, data regions are separated by alignment patterns. The data region, or set of data regions and alignment patterns, is surrounded by a finder pattern, and this shall in turn be surrounded on all four sides by a quiet zone border. Figure 1 illustrates two representations of a Data Matrix symbol, dark on light and reflectance reversal.

a) Data Matrix, dark on light

b) Data Matrix, light on dark (reflectance reversal)

Figure 1 — Data Matrix "A1B2C3D4E5F6G7H8I9J0K1L2"

6.3.2 Finder pattern

The finder pattern is a perimeter to the data region and is one module wide. Two adjacent sides, the left and lower sides, forming the L boundary, are solid lines; these are used primarily to determine physical size, orientation and symbol distortion. The two opposite sides are made up of alternating dark and light modules. These are used primarily to define the cell structure of the symbol, but also can assist in determining physical size and distortion. The extent of the quiet zone is indicated by the corner marks in Figure 1.

6.3.3 Symbol sizes and capacities

Data Matrix symbols have an even number of rows and an even number of columns. Some symbols are square with sizes from 10 x 10 to 144 x 144 not including quiet zones. Some symbols are rectangular with sizes from 8 x 18 to 16 x 48 not including quiet zones. For all Data Matrix code symbols, the upper right corner module has the opposite reflectance state (i.e. light or dark) of the "L" finder pattern (see Figure 1). The complete attributes are given in Table 10.

7 Data Matrix code requirements

7.1 Encode procedure overview

7.1.1 General

Subclause 7.1 provides an overview of the encoding procedure. Following sections will provide more details. An encoding example is given in Annex I. The following steps convert user data to a Data Matrix code symbol.

7.1.2 Step 1: data encodation

As Data Matrix includes various encodation schemes that allow a defined set of characters to be converted into codewords more efficiently than the default scheme, analyse the data stream to identify the variety of different characters to be encoded. Insert additional codewords to switch between the encodation schemes and to perform other functions. Add pad characters as needed to fill the required number of codewords. If the user does not specify the matrix size, then choose the smallest size that accommodates the data. A complete list of matrix sizes is shown in Table 10.

Table 1 — Encodation schemes for Data Matrix code

Encodation scheme	Characters	Bits per data character
ASCII	double digit numerics	4
	ASCII values 0 to 127	8
	Extended ASCII values 128 to 255	16
C40	Upper case alphanumeric	5,33
	Lower case and special characters	10,66[a]
Text	Lower case alphanumeric	5,33
	Upper case and special characters	10,66[b]
X12	ANSI X12 EDI data set	5,33
EDIFACT	ASCII values 32 to 94	6
Base 256	All byte values 0 to 255	8

[a] Encoded as two C40 values as result of use of a shift character.

[b] Encoded as two Text values as result of use of a shift character.

7.1.3 Step 2: error checking and correcting codeword generation

For symbols with more than 255 codewords, sub-divide the codeword stream into interleaved blocks to enable the error correction algorithms to be processed as shown in Annex A. Generate the error correction codewords for each block. The result of this process expands the codeword stream by the number of error correction codewords. Place the error correction codewords after the data codewords.

7.1.4 Step 3: module placement in matrix

Place the codeword modules in the matrix. Insert the alignment pattern modules, if any, in the matrix. Add the finder pattern modules around the matrix.

7.2 Data encodation

7.2.1 Overview

The data may be encoded using any combination of six encodation schemes (see Table 1). ASCII encodation is the basic scheme. All other encodation schemes are invoked from ASCII encodation and return to this scheme. The compaction efficiencies given in Table 1 need to be interpreted carefully. The best scheme for a given set of data may not be the one with the fewest bits per data character. If the highest degree of compaction is required, account has to be taken of switching between encodation schemes and between code sets within an encodation scheme (see Annex J). It should also be noted that even if the number of codewords is minimised, the codeword stream sometimes needs to be expanded to fill a symbol. This fill process is done using pad characters.

7.2.2 Default character interpretation

The default character interpretation for character values 0 to 127 shall conform to ISO/IEC 646 IRV. The default character interpretation for character values 128 to 255 shall conform to ISO/IEC 8859-1. The graphical representation of data characters shown throughout this document complies with the default interpretation. This interpretation can be changed using ECI escape sequences, see 7.3. The default interpretation corresponds to ECI 000003.

7.2.3 ASCII encodation

ASCII encodation is the default set for the first symbol character in all symbol sizes. It encodes ASCII data, double density numeric data and symbology control characters. Symbology control characters include function characters, the pad character and the switches to other code sets. ASCII data is encoded as codewords 1 to 128 (ASCII value plus 1). Extended ASCII (data values 128 to 255) is encoded using the upper shift symbology control character (see 7.2.4.3). The digit pairs 00 to 99 are encoded with codewords 130 to 229 (numeric value plus 130). The ASCII code assignments are shown in Table 2.

NOTE ASCII encodation is the name of the character set in Data Matrix. It is not to be confused with the ASCII character set (ISO/IEC 646 IRV).

Table 2 — ASCII encodation values

Codeword	Data or function
0	Not to be used in ASCII encodation
1 to 128	ASCII data (ASCII value + 1)
129	Pad
130 to 229	2-digit data 00 to 99 (Numeric Value + 130)
230	Latch to C40 encodation
231	Latch to Base 256 encodation
232	FNC1
233	Structured Append
234	Reader Programming
235	Upper Shift (shift to Extended ASCII)
236	05 Macro

Table 2 *(continued)*

Codeword	Data or function
237	06 Macro
238	Latch to ANSI X12 encodation
239	Latch to Text encodation
240	Latch to EDIFACT encodation
241	ECI Character
242 to 255	Not to be used in ASCII encodation

7.2.4 Symbology control characters

7.2.4.1 General

Data Matrix symbols have several special symbology control characters, which have particular significance to the encodation scheme. These characters shall be used to instruct the decoder to perform certain functions or to send specific data to the host computer as described in 7.2.4.2 to 7.2.4.10. These symbology control characters, with the exception of values from 242 through 255, are found in the ASCII encodation (see Table 2).

7.2.4.2 Latch characters

A latch character shall be used to switch from ASCII encodation to one of the other encodation schemes. All codewords which follow a latch character shall be compacted according to the new encodation scheme. The encodation schemes have different methods for returning to the ASCII encodation.

7.2.4.3 Upper Shift character

The Upper Shift character is used in combination with an ASCII value (1 to 128) to encode an extended ASCII character (129 to 255). An extended ASCII character encoded in the ASCII, C40, or Text encodation scheme requires a preceding Upper Shift character and the extended ASCII character value decreased by 128 is then encoded according to the rules of the encodation scheme. In ASCII encodation, the Upper Shift character is represented by codeword 235. The reduced data value (i.e. ASCII value minus 128) is transformed into its codeword value by adding 1. For example, to encode ¥ (Yen currency symbol) (ASCII value 165), an upper shift character (codeword 235) is followed by value 37 (165 to 128), which is encoded as codeword 38. If there are long data strings of characters from the extended ASCII range, a latch to Base 256 encodation should be more efficient.

7.2.4.4 Pad character

If the encoded data, irrespective of the encodation scheme in force, does not fill the data capacity of the symbol, pad characters (value 129 in ASCII encodation) shall be added to fill the remaining data capacity of the symbol. The pad characters shall only be used for this purpose. Before inserting pad characters, it is necessary to return to ASCII encodation if in any other encodation mode. The 253-State pattern randomising algorithm shall be applied to the pad characters starting at the second pad character as specified in Annex B.

7.2.4.5 ECI character

An ECI character is used to change from the default interpretation used to encode data. The ECI protocol is common across a number of symbologies and its application to Data Matrix is defined more fully in 7.3. The ECI character shall be followed by one, two or three codewords which identify the ECI being invoked. The new ECI remains in place until the end of the encoded data, or until another ECI character is used to invoke another interpretation. See also Annex M.

7.2.4.6 Shift characters in C40 and Text encodation

In C40 and Text encodation, three special characters, called shift characters, are used as a prefix to one of 40 values to encode about three quarters of the ASCII characters. This allows the remaining ASCII characters to be encoded in a more condensed way with single values.

7.2.4.7 FNC1 alternate data type identifier

To encode data to conform to specific industry standards as authorised by AIM Inc., a FNC1 character shall appear in the first or second symbol character position (or in the fifth or sixth data positions of the first symbol of Structured Append). FNC1 encoded in any other position is used as a field separator and shall be transmitted as $^{G}{}_{S}$ control character (ASCII value 29).

7.2.4.8 Macro characters

Data Matrix provides a means of abbreviating an industry specific header and trailer in one symbol character. This feature exists to reduce the number of symbol characters needed to encode data in a symbol using certain structured formats. A Macro character shall be in the first character position of a symbol. They shall not be used in conjunction with Structured Append and their functions are defined in Table 3. The header shall be transmitted as a prefix to the data stream and the trailer shall be transmitted as a suffix to the data stream. The symbology identifier, if used, shall precede the header.

Table 3 — Macro functions

Macro codeword	Name	Interpretation	
		Header	Trailer
236	05 Macro	[)>$^{R}{}_{S}$05$^{G}{}_{S}$	$^{R}{}_{S}$ $^{E}O_{T}$
237	06 Macro	[)>$^{R}{}_{S}$06$^{G}{}_{S}$	$^{R}{}_{S}$ $^{E}O_{T}$

7.2.4.9 Structured Append character

A Structured Append character is used to indicate that the symbol is part of a Structured Append sequence according to the rules defined in 7.5.

7.2.4.10 Reader Programming character

A Reader Programming character indicates that the symbol encodes a message used to program the reader system. The Reader Programming character shall appear as the first codeword of the symbol and Reader Programming shall not be used with Structured Append.

7.2.5 C40 encodation

7.2.5.1 General

The C40 encodation scheme is designed to optimise the encoding of upper-case alphabetic and numeric characters but also enables other characters to be encoded by the use of shift characters in conjunction with the data character.

C40 characters are partitioned into 4 subsets. Characters of the first set, called the basic set, are the three special shift characters, the space character, and the ASCII characters A to Z and 0 to 9. They are assigned to a single C40 values. Characters of the other sets shall be assigned to one of the three shift characters, pointing to one of the 3 remaining subset, followed by one of the C40 values (see Table C.1).

As a first stage, each data character is converted into a single C40 value or a pair of C40 values. The complete string of C40 values is then decomposed into groups of three values (special rules apply if one or two values remain at the end, see 7.2.5.3). Each triplet (*C1*, *C2*, *C3*) is then encoded into a 16-bit value according to the formula: (1600 * *C1*) + (40 * *C2*) + *C3* + 1. Each 16-bit value is then separated into 2 codewords by taking the most significant 8 bits and the least significant 8 bits.

7.2.5.2 Switching to and from C40 encodation

It is possible to switch to C40 encodation from ASCII encodation using the appropriate latch codeword (230). Codeword 254 immediately following a pair of codewords in C40 encodation acts as an unlatch codeword to switch back to ASCII encodation. Otherwise, the C40 encodation remains in effect to the end of the data encoded in the symbol.

7.2.5.3 C40 encodation rules

Each pair of codewords represents a 16-bit value where the first codeword represents the most significant 8 bits. Three C40 values (*C1*, *C2*, *C3*) shall be encoded as:

$$(1\,600 * C1) + (40 * C2) + C3 + 1$$

which produces a value from 1 to 64 000. Table 4 illustrates three C40 values compacted into two codewords. Characters in the Shift 1, Shift 2 and Shift 3 sets shall be encoded by first encoding the appropriate shift character, and then the C40 value for the data. C40 encodation may be in effect at the end of the symbol's codewords which encode data.

The following rules apply when only one or two symbol characters remain in the symbol before the start of the error correction codewords:

a) If two symbol characters remain and three C40 values remain to be encoded (which may include both data and shift characters) encode the three C40 values in the last two symbol characters. A final unlatch codeword is not required.

b) If two symbol characters remain and two C40 values remain to be encoded (the first C40 value may be a shift or data character but the second shall represent a data character) append a “Shift 1” character (C40 value 0) to the input to increase to three characters so that it can be processed normally as with any three character C40 input. A final unlatch codeword again is not required.

c) If two symbol characters remain and only one C40 value (data character) remains to be encoded, the first symbol character is encoded as an unlatch character and the last symbol character is encoded with the data character using the ASCII encodation scheme.

d) If one symbol character remains and one C40 value (data character) remains to be encoded, the last symbol character is encoded with the data character using the ASCII encodation scheme. The unlatch character is not encoded, but is assumed, before the last symbol character.

   In all other cases, either an unlatch character is used to exit the C40 encodation scheme before the end of the symbol, or a larger symbol size is required to encode the data.

Table 4 — Example of C40 encoding

Data characters	AIM
C40 values	14, 22, 26
Calculate 16-bit value	(1600 * 14) + (40 * 22) + 26 + 1 = 23307
1st codeword: (16-bit value) div 256	23 307 div 256 = 91
2nd codeword: (16-bit value) mod 256	23 307 mod 256 = 11
Codewords	91, 11

7.2.5.4 Use of Upper Shift with C40

In C40 encodation the Upper Shift character is not a symbology function character but a shift within the encodation set. When a data character from the extended ASCII character range is encountered, three or four values in C40 encodation need to be encoded according to the following rule:

IF [ASCII value − 128] is in the Basic Set then:

[1(Shift 2)] [30(Upper Shift)] [V(ASCII value – 128)]

ELSE

[1(Shift 2)] [30(Upper Shift)] [0, 1, or 2(Shift 1, 2, or 3)] [V(ASCII value – 128)]

In the rule the number in [ ] equates to the C40 values from Table C.1; V has been used to indicate the appropriate C40 value.

7.2.6 Text encodation

7.2.6.1 General

Text encodation is designed to encode normal printed text, which is predominantly lowercase characters. It is similar in structure to the C40 encodation set, except that lowercase alphabetic characters are directly encoded (i.e. without using a shift). Upper-case alphabetic characters are preceded by a Shift 3. The full Text encodation character set assignments shall be as shown in Table C.2.

7.2.6.2 Switching to and from Text encodation

It is possible to switch to Text encodation from ASCII encodation using the appropriate latch codeword (239). Codeword 254 immediately following a pair of codewords in text encodation acts as an unlatch codeword to switch back to ASCII encodation. Otherwise, the Text encodation remains in effect to the end of the data encoded in the symbol.

7.2.6.3 Text encodation rules

The rules for C40 encodation apply.

7.2.7 ANSI X12 encodation

7.2.7.1 General

ANSI X12 encodation is used to encode the standard ANSI X12 electronic data interchange characters, which are compacted three data characters to two codewords in a manner similar to C40 encodation. It encodes upper-case alphabetic characters, numerics, space and the three standard ANSI X12 terminator and separator characters. The ANSI X12 code assignments are shown in Table 5. There are no shift characters in the ANSI X12 encodation set.

Table 5 — ANSI X12 encodation set

X12 value	Encoded characters	ASCII values
0	X12 segment terminator <CR>	13
1	X12 segment separator *	42
2	X12 sub-element separator >	62
3	space	32
4 to 13	0 to 9	48 to 57
14 to 39	A to Z	65 to 90

7.2.7.2 Switching to and from ANSI X12 encodation

It is possible to switch to ANSI X12 encodation from ASCII encodation using the appropriate latch codeword (238). Codeword 254 immediately following a pair of codewords in ANSI X12 encodation acts as an unlatch codeword to switch back to ASCII encodation. Otherwise, the ANSI X12 encodation remains in effect to the end of the data encoded in the symbol.

7.2.7.3 ANSI X12 encodation rules

The rules of C40 encodation apply. The exception is at the end of encoding ANSI X12 data. If the data characters do not fully utilise pairs of codewords, then following the last complete pair of codewords switch to ASCII using codeword 254 and continue using ASCII encodation, except when a single symbol character is left at the end before the first error correction character. This single symbol character uses the ASCII encodation scheme without requiring an unlatch codeword.

7.2.8 EDIFACT encodation

7.2.8.1 General

The EDIFACT encodation scheme includes 63 ASCII values (values from 32 to 94) plus an unlatch character (binary 011111) to return to ASCII encodation. EDIFACT encodation encodes four data characters in three codewords. It includes all the numeric, alphabetic and punctuation characters defined in the EDIFACT Level A character set without any of the shifts required in C40 encodation.

7.2.8.2 Switching to and from EDIFACT encodation

It is possible to switch to EDIFACT encodation from ASCII encodation using the appropriate latch codeword (240). The unlatch character in EDIFACT encodation shall be used as a terminator at the end of EDIFACT encodation, which reverts to ASCII encodation.

7.2.8.3 EDIFACT encodation Rules

The EDIFACT encodation character set is defined in Table C.3. There is a simple relationship between the 6-bit EDIFACT value and the ASCII 8-bit byte. The leading two bits of the 8-bit byte are ignored to create the EDIFACT 6-bit value, as illustrated in Table 6. Strings of four EDIFACT characters are encoded in three codewords. For a simple encodation process, the leading two bits of the 8-bit byte are removed. The remaining 6-bit byte is the EDIFACT value and shall be directly encoded into the codeword as illustrated in Table 7. When EDIFACT encodation is terminated with the unlatch character, any remaining bits left in the single symbol character shall be filled with zeros. ASCII encodation starts with the next symbol character. If EDIFACT encodation is in effect at the end of the symbol before the first error correction character, and only one or two codewords remain after the last EDIFACT codeword triplet, these remaining codewords shall be encoded in ASCII encodation without requiring an unlatch character.

Table 6 — Relationship between the EDIFACT value and the 8-bit byte value

Data character	ASCII		EDIFACT value
	Decimal value	8-bit binary value	
A	65	01000001	000001
9	57	00111001	111001

NOTE During the decode process, if the leading (6th) bit is 1, the bits 00 are prefixed to create the 8-bit byte. If the leading (6th) bit is 0, the bits 01 are prefixed to create the 8-bit byte. The exception to this is the EDIFACT value 011111 which is the symbology control unlatch character to return to ASCII encodation.

Table 7 — Example of EDIFACT encodation

Data characters	D			A			T			A		
Binary values (Table C.3)	00	01	00	00	00	01	01	01	00	00	00	01
Divide into 3 8-bit bytes	00	01	00	00	00	01	01	01	00	00	00	01
Codeword values	16				21				1			

7.2.9 Base 256 encodation

7.2.9.1 General

The Base 256 encodation scheme shall be used to encode any 8-bit byte data, including extended channel interpretations and binary data. The default interpretation is defined in 7.2.2. The 255-State pattern randomising algorithm shall be applied to each Base 256 sequence within the encoded data (see B.2). It starts after the latch to Base 256 encodation and ends at the last character specified by the Base 256 field length.

7.2.9.2 Switching to and from Base 256 encodation

It is possible to switch to Base 256 encodation from ASCII encodation using the appropriate latch codeword (231). At the end of Base 256 encodation, encodation automatically reverts to ASCII encodation. The appropriate ECI, if other than the default, shall be invoked prior to switching. The ECI sequence need not occur immediately before switching to Base 256 encodation.

7.2.9.3 Base 256 encodation rules

After switching to Base 256 encodation, the first one (*d1*) or two (*d1*, *d2*) codewords define the data field length in bytes. Table 8 specifies how the field length is defined. Thereafter, all encodation shall be of the byte values.

Table 8 — Base 256 field length

Field length	**Values of *d1*, *d2***	**Permitted values of *d***
Remainder of symbol	*d1* = 0	*d1* = 0
1 to 249	*d1* = length	*d1* = 1 to 249
250 to 1555	*d1* = (length DIV 250) + 249	*d1* = 250 to 255
	d2 = length MOD 250	*d2* = 0 to 249

7.3 ECI

7.3.1 General

The ECI protocol allows the output data stream to have interpretations different from that of the default character set. The ECI protocol is defined consistently across a number of symbologies. Four broad types of interpretations are supported in Data Matrix:

a) international character sets (or code pages);

b) general purpose interpretations such as encryption and compaction;

c) user defined interpretations for closed systems;

d) control information for structured append in unbuffered mode.

The ECI protocol is fully specified in Reference [1]. The protocol provides a consistent method to specify particular interpretations on byte values before printing and after decoding. The ECI is identified by a 6-digit number which is encoded in the Data Matrix symbol by the ECI character followed by one to three codewords. Specific interpretations are listed in AIM Inc. Extended Channel Interpretations Character Set Register. The ECI can only be used with readers enabled to transmit the symbology identifiers. Readers that are not enabled to transmit the symbology identifier shall not transmit the data from any symbol containing an ECI. An exception can be made if the ECI(s) can be handled entirely within the reader.

A specified Extended Channel Interpretation may be invoked anywhere in the encoded message.

7.3.2 Encoding ECIs

The various encodation schemes of Data Matrix (defined in Table 1) may be applied under any of the ECIs. The ECI can only be invoked from ASCII encodation; once this has occurred, switching may take place between any of the encodation schemes. The encodation mode used is determined strictly by the 8-bit data values being encoded and does not depend on the Extended Channel Interpretation in force. For example, a sequence of values in the range 48 to 57 (decimal) would be most efficiently encoded in numeric mode even if they were not to be interpreted as numbers. The ECI assignment is invoked using codeword 241 (ECI character) in ASCII encodation. One, two, or three additional codewords are used to encode the ECI Assignment number. The encodation rules are defined in Table 9.

The following examples illustrate the encodation:

ECI = 015000

Codewords:

[241] [(15000 - 127) div 254 + 128] [(15000 - 127) mod 254 + 1]

= [241] [58 + 128] [141 + 1]

= [241] [186] [142]

ECI = 090000

Codewords:

[241] [(90000 - 16383) div 64516 + 192] [((90000 - 16383) div 254) mod 254 + 1] [(90000 - 16383) mod 254 + 1]

= [241] [1 + 192] [289 mod 254 + 1] [211 + 1]

= [241] [193] [36] [212]

Table 9 — Encoding ECI assignment numbers in Data Matrix code

ECI assignment value	Codeword sequence	Codeword values	Ranges
000000 to 000126	C_0	241	
	C_1	ECI_no + 1	C_1 = (1 to 127)
000127 to 016382	C_0	241	
	C_1	(ECI_no - 127) div 254 + 128	C_1 = (128 to 191)
	C_2	(ECI_no - 127) mod 254 + 1	C_2 = (1 to 254)
0016383 to 999999	C_0	241	
	C_1	(ECI_no - 16383) div 64516 +192	C_1 = (192 to 207)
	C_2	[(ECI_no - 16383) div 254] mod 254 + 1	C_2 = (1 to 254)
	C_3	(ECI_no - 16383) mod 254 + 1	C_3 = (1 to 254)

7.3.3 ECIs and Structured Append

ECIs may occur anywhere in the message encoded in a single or Structured Append (see 7.5) set of Data Matrix symbols. Any ECI invoked shall apply until the end of the encoded data, or until another ECI is encountered. Thus, the interpretation of the ECI may straddle two or more symbols.

7.3.4 Post-decode protocol

The protocol for transmitting ECI data shall be as defined in 12.5. When using ECIs, symbology identifiers (see 12.6) shall be fully implemented and the appropriate symbology identifier transmitted as a preamble.

7.4 Data Matrix symbol attributes

7.4.1 Symbol sizes and capacity

There are 24 square symbols and 6 rectangular symbols available in Data Matrix code. These are as specified in Table 10.

NOTE Additional rectangular symbol formats are specified in ISO/IEC 21471.

Table 10 — Data Matrix symbol attributes

Symbol size[a]		Data region		Mapping matrix size	Total codewords		Reed-Solomon block		Inter-leaved blocks	Maximum data capacity			% of codewords used for error correction	Max. correctable codewords Error/ erasure[b]
Row	Col	Size	No.		Data	Error	Data	Error		Num.	Alphanum.[d]	Byte		
10	10	8 x 8	1	8 x 8	3	5	3	5	1	6	3	1	62,5	2/0
12	12	10 x 10	1	10 x 10	5	7	5	7	1	10	6	3	58,3	3/0
14	14	12 x 12	1	12 x 12	8	10	8	10	1	16	10	6	55,6	5/7
16	16	14 x 14	1	14 x 14	12	12	12	12	1	24	16	10	50	6/9
18	18	16 x 16	1	16 x 16	18	14	18	14	1	36	25	16	43,8	7/11
20	20	18 x 18	1	18 x 18	22	18	22	18	1	44	31	20	45	9/15
22	22	20 x 20	1	20 x 20	30	20	30	20	1	60	43	28	40	10/17
24	24	22 x 22	1	22 x 22	36	24	36	24	1	72	52	34	40	12/21
26	26	24 x 24	1	24 x 24	44	28	44	28	1	88	64	42	38,9	14/25
32	32	14 x 14	4	28 x 28	62	36	62	36	1	124	91	60	36,7	18/33
36	36	16 x 16	4	32 x 32	86	42	86	42	1	172	127	84	32,8	21/39
40	40	18 x 18	4	36 x 36	114	48	114	48	1	228	169	112	29,6	24/45
44	44	20 x 20	4	40 x 40	144	56	144	56	1	288	214	142	28	28/53
48	48	22 x 22	4	44 x 44	174	68	174	68	1	348	259	172	28,1	34/65
52	52	24 x 24	4	48 x 48	204	84	102	42	2	408	304	202	29,2	42/78
64	64	14 x 14	16	56 x 56	280	112	140	56	2	560	418	277	28,6	56/106
72	72	16 x 16	16	64 x 64	368	144	92	36	4	736	550	365	28,1	72/132
80	80	18 x 18	16	72 x 72	456	192	114	48	4	912	682	453	29,6	96/180
88	88	20 x 20	16	80 x 80	576	224	144	56	4	1 152	862	573	28	112/212
96	96	22 x 22	16	88 x 88	696	272	174	68	4	1 392	1 042	693	28,1	136/260
104	104	24 x 24	16	96 x 96	816	336	136	56	6	1 632	1 222	813	29,2	168/318
120	120	18 x 18	36	108 x 108	1 050	408	175	68	6	2 100	1 573	1 047	28	204/390
132	132	20 x 20	36	120 x 120	1 304	496	163	62	8	2 608	1 954	1 301	27,6	248/472
144	144	22 x 22	36	132 x 132	1 558	620	156	62	8[c]	3 116	2 335	1 555	28,5	310/590
							155	62	2[c]					
Rectangular Symbols														
8	18	6 x 16	1	6 x 16	5	7	5	7	1	10	6	3	58,3	3/0
8	32	6 x 14	2	6 x 28	10	11	10	11	1	20	13	8	52,4	5/0
12	26	10 x 24	1	10 x 24	16	14	16	14	1	32	22	14	46,7	7/11
12	36	10 x 16	2	10 x 32	22	18	22	18	1	44	31	20	45,0	9/15
16	36	14 x 16	2	14 x 32	32	24	32	24	1	64	46	30	42,9	12/21
16	48	14 x 22	2	14 x 44	49	28	49	28	1	98	72	47	36,4	14/25

a Symbol size does not include quiet zones.

b See 7.6.3.

c In the largest symbol (144 x 144), the first eight Reed-Solomon blocks are 218 codewords long encoding 156 data codewords, and the last two blocks encode 217 codewords (155 data codewords). All the blocks have 62 error correction codewords.

d Based on text or C40 encoding without switching or shifting; for other encoding schemes, this value may vary depending on the mix and grouping of character sets.

7.4.2 Insertion of Alignment Patterns into larger symbols

As shown in Table 10, square symbols 32 x 32 and larger and four rectangular symbols (8 x 32, 12 x 36, 16 x 36, and 16 x 48) have two or more data regions. These data regions shall be bound by alignment patterns in accordance with Annex D. The square symbols are divided into 4, 16 or 36 data regions (as illustrated in Figures D.1, D.2 and D.3). The rectangular symbols are divided into two data regions (as illustrated in Figure D.4). The alternating dark modules of the alignment pattern shall be to the top and right of a data region and identify the even columns and rows.

7.5 Structured Append

7.5.1 Basic principles

Up to 16 Data Matrix symbols may be appended in a structured format. If a symbol is part of a Structured Append, this is indicated by codeword 233 in the first symbol character position. This is immediately followed by three structured append codewords. The first codeword is the symbol sequence indicator. The second and third codewords are the file identification.

7.5.2 Symbol sequence indicator

This codeword indicates the position of the symbol within the set (up to 16) of Data Matrix symbols in the Structured Append format in the form *m* of *n* symbols. The first 4 bits of this codeword identify the position of the particular symbol as the binary value of (m – 1). The last 4 bits identify the total number of the symbols to be concatenated in the Structured Append format as the binary value of (17 – n). The 4-bit patterns shall conform with those defined in Table 11.

Table 11 — Structured Append symbol position bits

Symbol position	Bits 1234	Total number of symbols	Bits 5678
1	0000		
2	0001	2	1111
3	0010	3	1110
4	0011	4	1101
5	0100	5	1100
6	0101	6	1011
7	0110	7	1010
8	0111	8	1001
9	1000	9	1000
10	1001	10	0111
11	1010	11	0110
12	1011	12	0101
13	1100	13	0100
14	1101	14	0011
15	1110	15	0010
16	1111	16	0001

For example, the 3rd symbol of a set of 7 shall be encoded thus:

3rd position: 0010

Total 7 symbols: 1010

Bit pattern: 00101010

Codeword value: 42

7.5.3 File identification

The file identification is defined by the value of its two codewords. Each file identification codeword may have a value 1 to 254, allowing 64 516 different file identifications. The purpose of the file identification is to increase the probability that only logically linked symbols are processed as part of the same message.

7.5.4 FNC1 and Structured Append

If Structured Append is used in conjunction with FNC1 (see 7.2.4.7), the first four codewords shall be used for Structured Append and the fifth and sixth codewords are available for FNC1 usage. FNC1 shall not be repeated in these positions in the second and subsequent symbols, except when used as a field separator.

7.5.5 Buffered and unbuffered operation

The message within a Structured Append sequence can be buffered in the reader in its entirety and transmitted after all of the symbols have been read. Alternatively, the reader may transmit the decoded data in each symbol as it is read. In this unbuffered operation, the ECI protocol for structured append (specified in AIM ITS/04-001, Part 1) defines a control block that shall be prefixed to the beginning of the data transmitted for each symbol.

7.6 Error detection and correction

7.6.1 Reed-Solomon error correction

Data Matrix symbols employ Reed-Solomon error correction. For Data Matrix symbols with less than 255 total codewords, the error correction codewords are calculated from data codewords with no interleaving. For Data Matrix symbols with more than 255 total codewords, the error correction codewords shall be calculated from data codewords with the interleaving procedure described in Annex A. Each Data Matrix symbol has a specific number of data and error correction codewords which are divided into a specific number of blocks, as defined in Table 10, and to which the interleaving procedure defined in Annex A is applied.

The polynomial arithmetic for Data Matrix shall be calculated using bit-wise modulo 2 arithmetic and byte-wise modulo 100101101 (decimal 301) arithmetic. This is a Galois field of 2^8 with 100101101 representing the field's prime modulus polynomial: $x^8 + x^5 + x^3 + x^2 + 1$. Sixteen different generator polynomials are used for generating the appropriate error correction codewords. These are given in Clause E.1.

NOTE An example implementation of the error correction algorithm is available in Reference [2].

7.6.2 Generating the error correction codewords

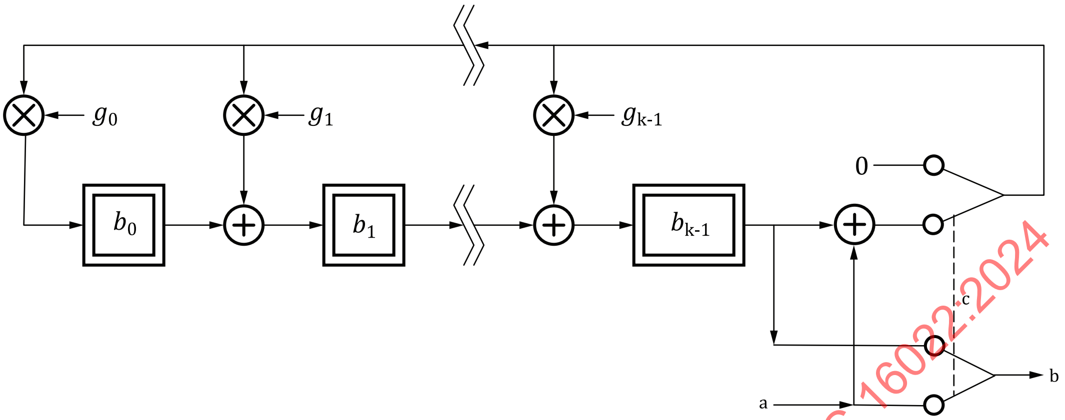
The error correction codewords are the remainder after dividing the data codewords by a polynomial $g(x)$ used for Reed-Solomon codes (see Clause E.1).

If this calculation is performed by "long division", the symbol data polynomial shall first be multiplied by x^k.

The data codewords are the coefficients of the terms of a polynomial with the coefficient of the highest term being the first data codeword and the lowest power term being the last data codeword before the first error correction codeword. The highest order coefficient of the remainder is the first error correction codeword and the zero power coefficient is the last error correction codeword and the last codeword. This can be implemented by using the division circuit as shown in Figure 2. The registers b_0 through b_{k-1} are initialised as zeros. There are two phases to generate the encoding. In the first phase, with the switch in the down position the data codewords are passed both to the output and the circuit. The first phase is complete after n clock pulses. In the second phase ($n + 1 \ldots n + k$ clock pulses), with the switch in the up position, the error correction codewords $\varepsilon_{k-1}, \ldots, \varepsilon_0$ are generated by flushing the registers in order while keeping the data input at 0. The codewords output from the shift register are in the order that they are to be placed in the symbol.

NOTE 1 If interleaving is used, the codewords will not be placed in consecutive symbol characters. (See Annex A).

NOTE 2 *n* and *k* are defined in Clause 4 as the number of data codewords and the number of error correction codewords, respectively.



Key

⊕ GF(256) addition

⊗ GF(256) multiplication

a Input.

b Output.

c Switch.

Figure 2 — Error correction codeword encoding circuit

7.6.3 Error correction capacity

The error correction codewords can correct two types of erroneous codewords: erasures (erroneous codewords at known locations) and errors (erroneous codewords at unknown locations). An erasure is an unscanned or undecodable symbol character. An error is a misdecoded symbol character. The number of erasures and errors that can be corrected is given by Formula (1):

$$e + 2t \leq d - p \quad (1)$$

where

- e is the number of erasures;
- t is the number of errors;
- d is the number of error correction codewords;
- p is the number of codewords reserved for error detection.

In the general case, $p = 0$. However, if most of the error correction capacity is used to correct erasures, then the possibility of an undetected error is increased. Whenever the number of erasures is more than half the number of error correction codewords, $p = 3$. For small symbols (10 x 10, 12 x 12, 8 x 18, and 8 x 32), erasure correction should not be used ($e = 0$ and $p = 1$).

7.7 Symbol construction

7.7.1 General

Given the codeword sequence obtained in the previous sections, a Data Matrix symbol is constructed using the following steps.

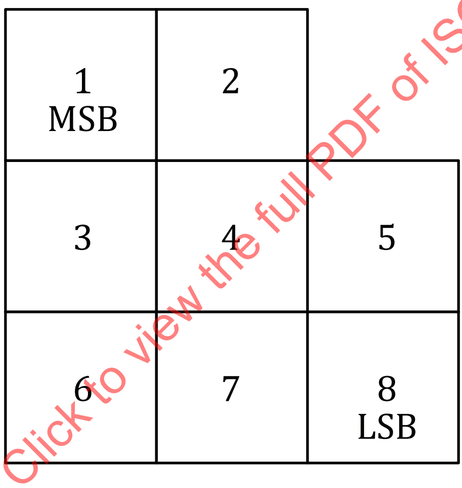
a) Place codeword modules in a mapping matrix.

b) Insert alignment pattern modules, if any.

c) Place finder modules along the perimeter.

7.7.2 Symbol character placement

Each symbol character shall be represented by eight modules which are nominally square in shape; each module represents a binary bit. A dark module is a one and a light module is a zero. The eight modules are in order from left to right and top to bottom to form a symbol character as shown in Figure 3. Because the symbol character shape defined in Figure 3 cannot be perfectly nested at the symbol boundary, some symbol characters are split into portions. Symbol character placement is defined in the C language program in Clause F.1, described in Clause F.2 and illustrated in Clause F.3 in Annex F. Symbol character placement as defined in Annex F shall be used.



Key

LSB least significant bit

MSB most significant bit

Figure 3 — Representation of a codeword in a symbol character

7.7.3 Alignment Pattern module placement

This step is only needed for larger matrices:

— square: 32 x 32, and

— larger rectangular: 8 x 32, 12 x 36 and larger.

The mapping matrix is sub-divided into data regions, of the sizes defined in Table 10, for the chosen symbol format. The data regions are separated from each other by two-module-wide alignment patterns. This will result in some of the symbol characters being split between two adjacent data regions. For square matrices, the alignment patterns are placed between the data regions horizontally and vertically in pairs with a total alignment pattern count of 2, 6 or 10 as shown in Figures D.1 to D.3. For rectangular matrices, only a single vertical alignment pattern is placed between the data regions as shown in Figure D.4.

7.7.4 Finder Pattern module placement

Modules are placed along the perimeter of the matrix to construct the finder pattern as described in 6.3.2.

8 Symbol dimensions

Data Matrix symbols shall conform to the following dimensions.

— *X* dimension: the width of a module shall be specified by the application, taking into account the scanning technology to be used, and the technology to produce the symbol.

— Finder pattern: the width of the finder pattern shall be equal to *X*.

— Alignment pattern: the width of the alignment pattern shall be equal to 2*X*.

— Quiet zone: the minimum quiet zone is equal to *X* on all four sides. For applications with moderate to excessive reflectance noise in close proximity to the symbol, a quiet zone of 2*X* to 4*X* is recommended.

9 Symbol quality

9.1 General

Data Matrix symbols shall be assessed for quality using the 2D matrix bar code symbol print quality specification defined in ISO/IEC 15415 or ISO/IEC 29158 in accordance with an applicable application specification, as augmented and modified below.

Some marking technologies can be unable to produce symbols conforming to this document without taking special precautions.

9.2 Symbol quality parameters

9.2.1 Fixed pattern damage

Annex G defines the measurement and grading basis for Fixed Pattern Damage.

9.2.2 Overall symbol grade

The grading method of ISO/IEC 15415 or ISO/IEC 29158 shall be used in accordance with an application specification.

9.2.3 Decode

The reference decode algorithm specified in this document shall be applied to determine the grade for Decode. A failure of the reference decode algorithm to successfully decode the symbol shall result in a grade of 0 for Decode. Reference decode constructs the actual sampling grid which is used to sample the modules of the symbol for the purpose of decoding. In case of an invalid encodation (e.g. 7.1, 7.2 and 7.3 used incorrectly) the symbol shall also result in a grade of 0 for Decode.

9.2.4 Grid non-uniformity

The ideal grid is calculated by using the four corner points in the surrounding fixed pattern /solid area of the sampling grid (see Figure 4) for each data region and subdividing it equally in both axes.

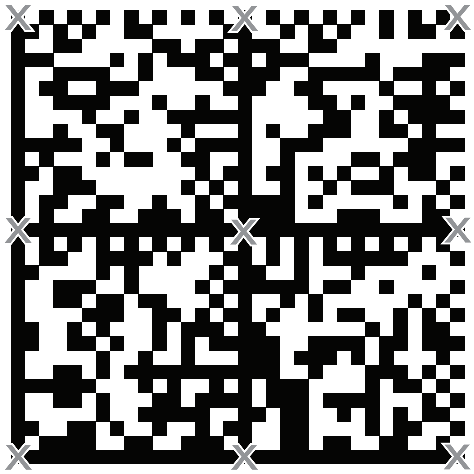


Figure 4 — Corner points of the ideal grid

9.3 Process control measurements

A variety of tools and methods can be used to perform useful measurements for monitoring and controlling the process of creating Data Matrix symbols. These techniques do not constitute a print quality check of the produced symbols. The method specified in 9.2 and in Annex G is the required method for assessing symbol print quality. However, other techniques individually and collectively yield good indications of whether the symbol print process is creating workable symbols.

10 Reference decode algorithm for Data Matrix

This reference decode algorithm finds a Data Matrix symbol in an image and decodes it.

a) Define measurement parameters and form a digitised image.

  1) Define a distance d_{min} which is 7,5 times the aperture diameter defined by the application. This will be the minimum length of the "L" pattern's side.

  2) Define a distance g_{max} which is 7,5 times the aperture diameter. This is the largest gap in the "L" finder that will be tolerated by the finder algorithm in step b).

  3) Define a distance m_{min} which is 1,25 times the aperture diameter. This would be the nominal minimum module size when the aperture size is 80 % of the symbol's *X* dimension.

  4) Form a black/white image using a threshold determined according to the method defined in ISO/IEC 15415.

b) Search horizontal and vertical scan lines for the two outside edges of the Data Matrix "L".

  1) Extend a scan line horizontally in both directions from the centre point of the image. Sample along the scan line. For each white/black or black/white transition found along the scan line resolved to the pixel boundary perform the following steps.

    i) Follow the edge upward sampling pixel by pixel until either it reaches a point 3,5 m_{min} distant from the intersection of the scan line and the edge starting point, or the edge turns back toward the intersection of the scan line and the edge – the starting point.

    ii) Follow the edge downward pixel by pixel until either it reaches a point 3,5 m_{min} distant from the intersection of the scan line and the edge starting point, or the edge turns back toward the intersection of the scan line and the edge – the starting point.

iii) If the upward edge reaches a point 3,5 m_{min} from the starting point, perform the following steps.

I) Plot a line A connecting the end points of the upward edge.

II) Test whether the intermediate edge points lie within 0,5 m_{min} from line A. If so, continue to step III. Otherwise proceed to step 1)iv) to follow the edge in the opposite direction.

III) Continue following the edge upward until the edge departs 0,5 m_{min} from line A or the edge turns back for a distance of at least 0,5 m_{min} . Back up along the edge to the closest edge point with an Euclidian distance greater than or equal to m_{min} from the last edge point along the edge before the departing point and save this as the edge end point. This edge point should be along the "L" candidate outside edge.

IV) Continue following the edge downward until the edge departs 0,5 m_{min} from line A or the edge turns back for a distance of at least 0,5 m_{min}. Back up along the edge to the closest edge point with a Euclidian distance greater than or equal to m_{min} from the last edge point along the edge before the departing point and save this as the edge end point. This edge point should be along the "L" candidate outside edge.

V) Calculate a new adjusted line A1 that is a "best fit" line to the edge in the two previous steps. The "best fit" line uses the linear regression algorithm (using the end points to select the proper dependent axis, i.e. if closer to horizontal, the dependent axis is x) applied to each point. The "best fit" line terminates lines at points p1 and p2 that are the points on the "best fit" line closest to the endpoints of the edge.

VI) Save the line A1 segment two end points, p1 and p2. Also save the colour of the left side of the edge viewed from p1 to p2.

iv) If step iii) failed or did not extend upward by 3,5 m_{min} in step iii)IV), test if the downward edge reaches a point 3,5 m_{min} from the starting point. If so, repeat the steps in iii) but with the downward edge.

v) If neither steps iii) or iv) were successful, test if both the upward and downward edges terminated at least 2 m_{min} from the starting point. If so, form an edge comprised of the appended 2 m_{min} length upward and downward edge segments and repeat the steps in iii) but with the appended edge.

vi) Proceed to and process the next transitions on the scan line, repeating from step i), until the boundary of the image is reached.

2) Extend a scan line vertically in both directions from the centre point of the image. Look for line segments using the same logic in step 1) above but following each edge transition first left and then right.

3) Search among the saved line A1 segments for pairs of line segments that meet the following four criteria.

i) If the two lines have the same p1 to p2 directions, verify that the closer of the interline p1 to p2 distances is less than g_{max}. If the two lines have opposite p1 to p2 directions, verify that the closer of the interline p1 to p1 or p2 to p2 distances is less than g_{max}.

ii) Verify that the two lines are co-linear within 5 degrees.

iii) Verify that the two lines have the same saved colour if their p1 to p2 directions are the same or that the saved colours are opposite if their p1 to p2 directions are opposite to each other.

iv) Form two temporary lines by extending each line to reach the point on the extension that is closest to the furthest end point of the other line segment. Verify that the two extended lines are separated by less than 0,5 m_{min} at any point between the two extended lines.

4) For each pair of lines meeting the criteria of step 3) above, replace the pair of line segments with a longer A1 line segment that is a "best fit" line to the four end points of the pair of shorter line

segments. Also save the colour of the left side of the edge of the new longer line viewed from its p1 endpoint to its p2 endpoint.

5) Repeat steps 3) and 4) until no more A1 line pairs can be combined.

6) Select line segments that are at least as long as d_{min}. Flag them as "L" side candidates.

7) Look for pairs of "L" side candidates that meet the following three criteria.

   i) Verify that the closest points on each line are separated by less than 1,5 g_{max}.

   ii) Verify that they are perpendicular within 5 degrees.

   iii) Verify that the same saved colour is on the inside of the "L" formed by the two lines. Note that if one or both lines extend past their intersection, then the two or four "L" patterns formed will need to be tested for matching colour and maintaining a minimum length of d_{min} for the truncated side or sides before they can become "L" candidates.

8) For each candidate "L" pair found in step 7) form an "L" candidate by extending the segments to their intersection point.

9) If the "L" candidate was formed from line segments with the colour white on the inside of the "L", form a colour inverted image to decode. Attempt to decode the symbol starting with the appropriate normal or inverted image starting from step d) below using each of the "L" candidates from step 8) as the "L" shaped finder. If none decode, proceed to step c).

c) Maintain the line A1 line segments and "L" side candidates from the previous steps. Continue searching for "L" candidates using horizontal and vertical scan lines offset from previous scan lines.

1) Using a new horizontal scan line 3 m_{min} above the centre horizontal scan line, repeat the process in step b)1), except starting from the offset from the centre point, and then b)3) through b)9). If there is no decode, proceed to the next step.

2) Using a new vertical scan line 3 m_{min} left of the centre vertical scan line, repeat the process in step b)2), except starting from the offset from the centre point, and then steps b)3) through b)9). If there is no decode, proceed to the next step.

3) Repeat step 1) above except using a new horizontal scan line 3 m_{min} below the centre horizontal scan line. If there is no decode, repeat step 2) above except using a new vertical scan line 3 m_{min} right of the centre vertical scan line. If there is no decode, proceed to step 4) below.

4) Continue processing horizontal and vertical scan lines as in steps 1) through 3) that are 3 m_{min} above, then left, then below, then right of the previously processed scan lines until either a symbol is decoded or the boundary of the image is reached.

d) First assume that the candidate area contains a square symbol. If the area fails to decode as a square symbol, then try to find and decode a rectangular symbol starting from procedure j). For a square symbol, first plot a normalised graph of transitions for the equal sides of the candidate area in order to find the alternating module finder pattern.

1) Project a line through the candidate area bisecting the interior angle of the two sides of the "L" found above as shown in Figure 5. Define the two equal areas formed by the bisecting line as the right side and the left side as viewed from the corner of the "L".

2) For each side, form a line called a “search line” between a point d_{min} distance from the corner along the “L” line, parallel to the other “L” side line, and extending to the bisecting line as shown in Figure 5.

3) Move each search line away from the corner of the “L” as shown in Figure 5, lengthening each line as it expands to span its two bounding lines, the “L” line and the bisecting line. Keep each search line parallel to the other “L” side line. As each side is moved by the size of an image pixel, count the number of black/white and white/black transitions, beginning and ending the count with transitions from the colour of the “L” side to the opposite colour. A transition from one colour to

the other is to be counted only when at the two adjacent parallel search lines (the previous and the next one), a transition of the same type (black to white or white to black) is detected at the corresponding position. Plot the number of transitions multiplied by the length of the longest "L" side divided by the current length of the search line measured between the two bounding lines, as indicated in Formula (2):

$$T = \frac{N \times L_{max}}{L_s} \quad (2)$$

where

N is the number of transitions;

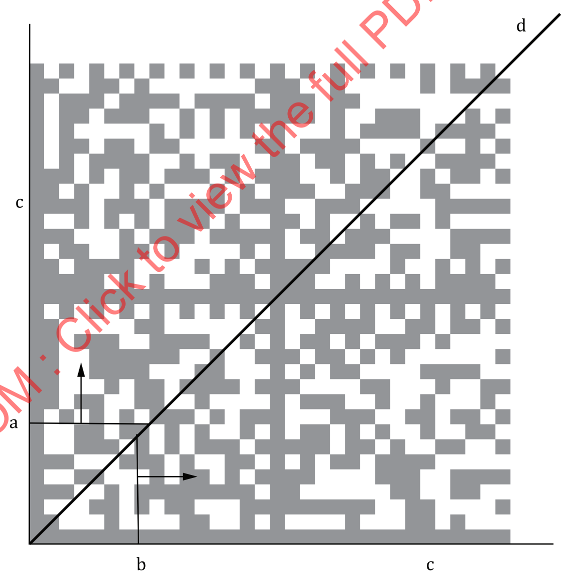
L_{max} is the "L" maximum line length;

L_s is the search line length;

T is the normalized transition count.

This formula normalises T to keep it from increasing because the line lengthens.

Continue to calculate the T values until the search line is longer than the longest axis of the candidate area plus 50 %.



a Left search line.

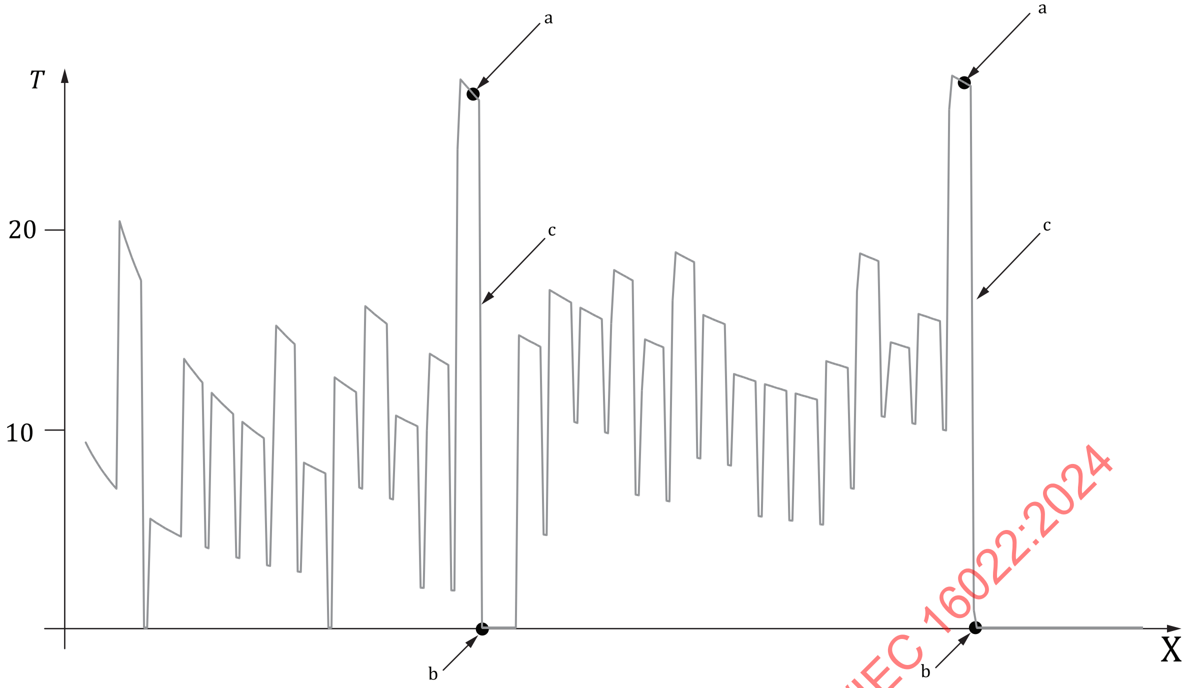
b Right search line.

c "L" side.

d Bisecting line.

Figure 5 — Expanding search lines

4) Form a plot of the T values for each side, where the Y-axis is the T value and the X-axis is the search line's distance from the corner of the "L". A sample plot is shown in Figure 6.



Key

X distance from L corner

T T values

[a] Peak.

[b] Valley.

[c] Candidate peak to valley descenders.

Figure 6 — Example plot of T as the search line expands

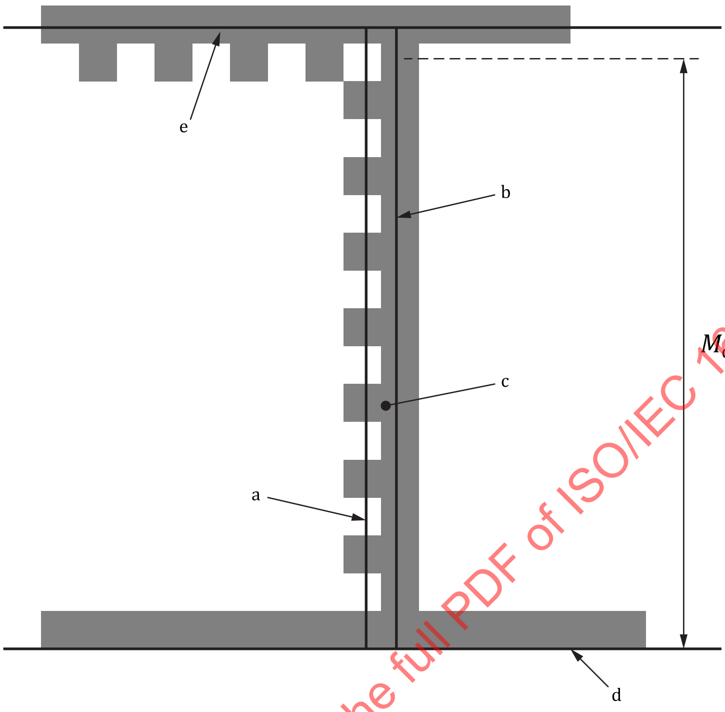
5) Starting from the *T* value with the smallest *X* in the right side's plot and then increasing *X*, find the first instance of a T_S value (T_S = maximum of zero and *T* – 1) that is less than 25 % of the preceding local maximum *T* value, provided that *T* value is greater than 1. Increment this *X* value until the number of transitions stops decreasing. If the number of transitions does not increase, increment the *X* value once more. Refer to this *X* value as the valley. Increment the local maximum's *X* value until the number of transitions decreases and refer to this *X* as the peak. Refer to the average of the peak and valley *X* values as the descending line *X* value. The search line at the peak may correspond to an alternating finder pattern side. At the valley, the search line may correspond to the solid dark interior line or a light quiet zone.

6) Find the peak and valley in the left side's plot whose descending line *X* value most closely matches the right peak and valley's descending line *X* value. If returning to this step from a later step, consider additional left peaks and valleys, ordered in terms of how closely they match the right peak and valley. However, any left peak and valley under consideration shall be checked to ensure that the absolute difference between the right and left peak *X* values is less than 15 % of the average of the two peak *X* values and that the absolute difference between the right and left valley *X* values is less than 15 % of the average of the two valley *X* values. The 15 % specifies the maximum allowed foreshortening.

7) The right side's valley search line, the left side's valley search line, and the two sides of the "L" outline a possible symbol's data region. Process the data region according to step e). If the decode fails, find the next left peak and valley from step d)6). Once all left peaks and valleys have been discarded, discard the right side peak and valley and continue searching from step d)5) for the next right peak and valley.

e) For each of the two sides of the alternating pattern, find the line passing through the centre of the alternating light and dark modules as follows.

1) For each side, form a rectangular region bounded by the side's peak and valley search lines as the longer two sides of the rectangle, and the "L" side and the other side's valley search line as the shorter two sides, as shown in Figure 7.



a Peak line.

b Valley line.

c Rectangular region.

d L boundary.

e Valley line from the other side.

Figure 7 — Rectangular region construction

2) Within the rectangular region, find the best fit to the outer boundary as follows.

i) Traverse all test lines starting with and parallel to the valley line and continuing until the peak line is reached, looking for transitions to the opposite colour normally orthogonal to the test line. Select only transitions that are either dark to light or light to dark where the first colour matches the predominate colour of the image along the valley line.

ii) Create a graph that plots the distance from the peak line to the first transition found at each position across the search lines. Where no transition was found, use zero for the distance.

For each segment of this graph that is bounded by zero on both sides:

a) find the point on this graph that is in the centre of this segment and the edge point corresponding to this point on the graph;

b) find all transition points that are within a radius R from the point on the edge, where R is 25 % of the distance between the two zero points on the graph;

c) collect all these points found and continue to the next segment.

iii) Calculate a "best fit line" with linear regression using the points collected above. This line should pass along the outside of the alternating pattern, shown as "best fit line" in Figure 8.

3) For each side, construct a line parallel to the step e)2) line which is offset toward the "L" corner by the perpendicular distance from the "L" corner to the peak search line divided by twice the number of transitions in the peak search line plus one, according to Formula (3):

$$O = d / ((n+1) * 2) \tag{3}$$

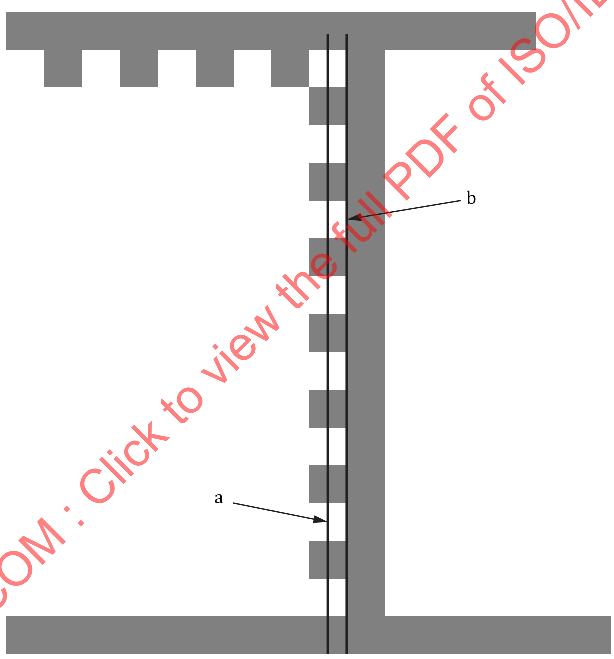
where

O is the offset

d is the distance to the peak line

n is the number of transitions

Each of the two constructed lines should correspond to the mid-line of the alternating module pattern on that side, see Figure 8.



a Alternating pattern module mid-line.

b Best fit line.

Figure 8 — Alternating pattern module mid-line

f) For each side, measure the edge-to-edge distances in the alternating pattern as follows.

1) Bound the alternating pattern mid-line constructed in step e)3) by the adjacent "L" line and the other alternating pattern mid-line from step e)3). Call the length of this line M_d (see Figure 7).

2) Along the bounded mid-line, measure the edge-to-edge distances between all the similar edges of all two-element pairs, i.e. dark/light and light/dark element pairs. Begin and end the edge-to-edge measurements with edges transitioning from the "L" colour to the opposite colour.

3) Select the median edge-to-edge measurement and set the current edge-to-edge measurement estimate, *EE_Dist*, to the median measurement.

4) Discard all element pairs with edge-to-edge measurements that differ more than 25 % from *EE_Dist*.

g) For each side, find the centre points of the alternating pattern modules:

1) Using the remaining element pair measurements from f)4), calculate the average ink spread (vertical or horizontal depending on the segment side) by the average of the element pair's ink spread, where "bar" is the dark element width and "space" is the light element width in a remaining element pair, according to Formula (4):

$$i = \text{Average}((w_b - ((w_b + w_s)/2))/((w_b + w_s)/2)) \tag{4}$$

where

i is the ink spread

w_b is the dark element (bar) width

w_s is the light element (space) width.

2) Calculate the centre of the bar in the median element pair using the offset in Formula (5) into the bar from the outside edge of the bar in the median pair:

$$o = (EE\_Dist * (1 + i))/4 \tag{5}$$

where

o is the offset

i is the ink spread.

If there is more than one median element pair, choose a single pair using the following process.

i) Order the edges (excluding the "L" finder edge) by their distance from the "L" finder edge. There is an odd number of these edges because the edges start and end on a dark to light transition going away from the "L" finder.

ii) Call the middle edge in the list the centre edge.

iii) Calculate the (odd number of) element pair edge-to-edge distances and find their median *EE_Dist*.

iv) Select the one or more element pairs with length *EE_Dist*.

v) Among those pairs identify the one or two element edge pairs that has an edge closest to the centre edge.

vi) If there is still a tie, take the element pair that has the outer edge of the bar closest to centre edge.

vii) If there is still a tie, take the element pair that has an inner edge closest to the "L" finder.

3) Starting from the centre of the bar in the median element pair from step f)3) proceed in the direction of the space in the element pair until reaching the end of the bounded mid-line, calculate each element's centre, shown by the speckled pattern in Figure 9, by the following steps.

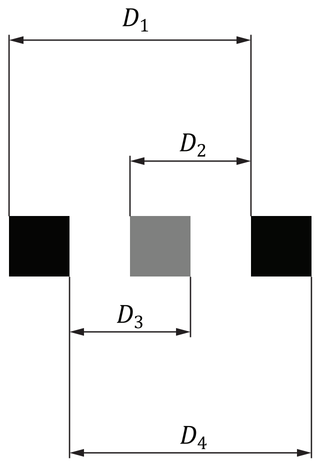


Figure 9 — Edge-to-edge measurements for finding an element centre

(While three bars and two spaces are shown in Figure 9, if a space is the element for which the centre is to be calculated, then the diagram would have three spaces instead of the bars and two bars instead of the spaces. For light elements adjacent to the element at the end of the mid-line, some of the *D1* through *D4* measurements are omitted as they would fall outside the symbol's or segment's measurable element boundaries.)

i) Calculate a point p1 along the mid-line which is *EE_Dist*/2 from the previously calculated element centre in the direction of the new element.

ii) Calculate d_1 through d_4 where:

d1 = *D1* / 2
d2 = *D2*
d3 = *D3*
d4 = *D4* / 2

iii) If one of the values d_1 through d_4 is within 25 % of *EE_Dist*, select the one which is closest to *EE_Dist*, and set the new *EE_Dist* to be the average of the current *EE_Dist* and the selected d_1 through d_4 distance.

I) If d_1 or d_4 are selected, select the corresponding *D1* or *D4* edge closest to the element, the centre of which is to be calculated. Offset this edge by *(i* / 2) * (*EE_Dist* / 2) in the appropriate direction (i.e. if *i* is positive, the offset will move the edge toward the space included in the distance *D1* or *D4*; if negative, the offset will move away from this space). Calculate a point p2 along the mid-line which is 0,75 times the selected d_1 or d_4 value from the offset edge and toward the element centre to be calculated.

II) If d_2 or d_3 are selected, select the corresponding *D2* or *D3* edge closest to the element the centre of which is to be calculated. Offset this edge by (*i* / 2) * (*EE_Dist* / 2) in the appropriate direction (i.e. if *i* is positive, the offset will move the edge toward the space included in the distance *D2* or D3; if negative, the offset will move away from this space). Calculate a point p2 along the mid-line which is 0,25 times the selected d_2 or d_3 value from the offset edge and toward the element centre to be calculated.

III) Set the element's centre as halfway between p1 and p2.

iv) Otherwise if none of the values *d1* through *d4* is within 25 % of *EE_Dist*, leave *EE_Dist* at its current value, use p1 as the new element's centre, and proceed to the next element.

4) To calculate the final element centre, advance from the current element centre by *EE_DIST* / 2.

5) Starting from the bar in the median element pair and proceeding in the opposite direction from step 3), until reaching the other end of the bounded mid-line, calculate each element's centre, following the procedures in step 3).

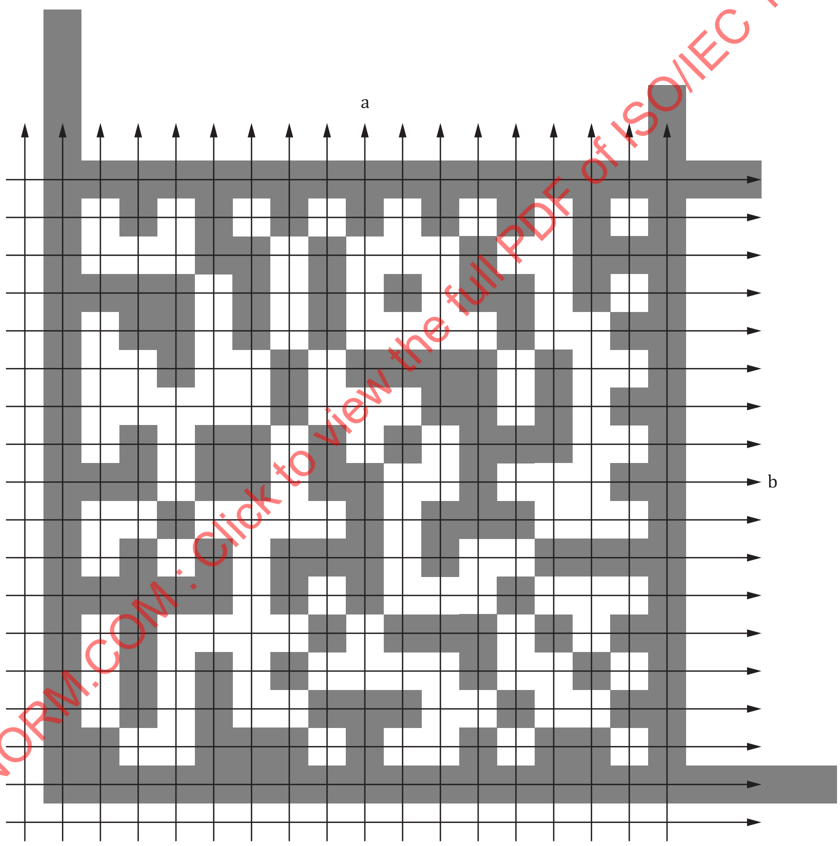
6) To calculate the final element centre, advance from the current element centre by *EE_DIST* / 2.

h) If the number of modules in each side does not correspond to a valid first region, continue searching from step d)6) for the next left peak and valley. Otherwise plot the data module sampling grid in the data region by extending the alternating pattern module centres as follows.

1) Extend each side's step e)3) mid-line and the opposite side's "L" line to form the vanishing point of the two nearly parallel or parallel extended lines.

2) Extend rays from each vanishing point passing through the step g) module centres of the nearly perpendicular step e)3) line.

3) The intersection of the two sets of nearly perpendicular rays should correspond to the centres of the data modules in the data region, as shown in Figure 10.



a To up side vanishing point.

b To right side vanishing point.

Figure 10 — Module sampling grid construction

i) Continue to fill in the remaining data regions as follows.

1) When a data region is processed, form a new "L" for the next data section to the "left" or "above" using one of the two following processes.

i) If the new data region is still bounded on one side by the original "L" from procedure b), repeat from procedure c) to process the new data region using the selected set of points from step e)2) and the set of points on the "L" from step b)2) which lie beyond the step e)2) line.

ii) If the new data region is bounded on two sides by data regions, repeat from procedure c) to process the new data region using the selected set of points from step e)2) for each data region which are adjacent and bound the new region on two sides.

2) If a data region does not match the number of modules in previously processed regions, trim the symbol to the largest number of regions which correspond to a legal symbol.

3) Decode the symbol with its one or more data regions starting with procedure k).

4) If the current data region exhausts its last peak and valley, revert to the previous data region and continue searching from step d)6) for the next left peak and valley in that data region.

j) Find the data sections of a rectangular symbol.

1) Check the lengths of the two "L" sides in the candidate pair and identify one as "longer" and the other as "shorter" within this candidate pair.

2) For each side of the "L" move a line perpendicular to the side and scanning along the length of the other side of the "L". Keep each search line parallel to the other "L" side line. As each side is moved by the size of an image pixel, count the number of black/white and white/black transitions, beginning and ending the count with transitions from the colour of the "L" side to the opposite colour. A transition from one colour to the other shall be counted only when the current search line as well as the search lines immediately above and below have the same colour, opposite to the previously counted transition colour. As each side is moved by a pixel, plot the number of transitions, T.

3) Continue until the parallel line moves further than the perpendicular leg of the "L" plus 50 % for the shorter side, or 25 %, if it is the longer side.

4) Starting from the origin of the plot, for each direction, find the first instance of a T_S value (T_S = maximum of zero and T − 1) value that is less than 15 % of the preceding local maximum T value, provided that T value is greater than 1. Increment this X value until the T value stops decreasing. If the T value does not increase, increment the X value once more. Refer to this X value as the valley. Increment the local maximum's X value until the T value decreases and refer to this X as the peak. Refer to the average of the peak and valley X Values as the descending line X value. The valley line at this point may form a side of a symbol or data region.

5) Find the alternating pattern lines for each side of the region similar to procedure e).

6) Plot the module sample grid in the data region or symbol as in procedures f), g), and h).

7) If the data region defined is not a valid rectangular symbol, try to form a new data region using further valid peak to valley plot transitions.

8) Process any additional regions as in procedure i).

9) If a valid data region or two regions are detected, attempt to decode the symbol as in procedures k) and l). If the region(s) were not valid or the decode fails, disregard the candidate area.

k) If the number of data modules is even or the symbol forms a valid rectangular symbol, decode the symbol using Reed-Solomon error correction as follows.

1) Sample the data modules at their predicted centres. Black at the centre is a one and white is a zero.

2) Convert the eight module samples in the defined codeword patterns into 8-bit symbol character values.

3) Apply Reed-Solomon error correction to the symbol character values.

4) Decode the symbol characters into data characters according to the specified encodation schemes.

NOTE 1 This reference decode algorithm was not designed to decode symbols composed of disconnected dots (e.g. dot peen). In this case, the reference decode algorithm fails by design. Applications can use the methods described in ISO/IEC 29158 to mitigate this limitation.

11 User guidelines

11.1 Human readable interpretation

Because Data Matrix symbols are capable of encoding thousands of characters, a human readable interpretation of the data characters can be impractical. As an alternative, descriptive text rather than the encoded text may accompany the symbol. The character size and font are not specified, and the message may be printed anywhere in the area surrounding the symbol. The human readable interpretation should not interfere with the symbol itself or the quiet zones.

11.2 Autodiscrimination capability

Data Matrix can be used in an autodiscrimination environment with a number of other symbologies (see Annex K).

11.3 System considerations

Data Matrix applications shall be viewed as a total system solution (see Annex L).

12 Transmitted data

12.1 General

This section describes the standard transmission protocol for compliant readers. These readers may be programmable to support other transmission options. All encoded data characters are included in the data transmission. The symbology control characters and error correction characters are not transmitted. More complex interpretations are addressed below.

12.2 Protocol for FNC1

When FNC1 appears in the first symbol character position (or in the fifth symbol character position of the first symbol of a Structured Append sequence), it shall signal that the data conforms to the GS1 Application Identifier standard format (see Reference [3]). FNC1 in any other later position in such symbols acts as a field separator. Transmission of symbology identifiers shall be enabled. The first FNC1 shall not be represented in the transmitted data, although its presence is indicated by the use of the appropriate option value (2) in the symbology identifier (see 12.6).

When used as a field separator, FNC1 shall be represented in the transmitted message by the ASCII character <$^{G}_{S}$> (ASCII value 29).

12.3 Protocol for FNC1 in the second position

When FNC1 is in the second symbol character position (or in the sixth symbol character position of the first symbol of a Structured Append sequence), it shall signal that the data conforms to a particular industry standard format. Transmission of symbology identifiers shall be enabled. The first FNC1 shall not be represented in the transmitted data, although its presence is indicated by the use of the appropriate option value (3) in the symbology identifier (see 12.6).

The data encoded in the first symbol character shall be transmitted as normal at the beginning of the data. When used as a field separator, FNC1 shall be represented in the transmitted message by the ASCII character <$^{G}{}_{S}$> (ASCII value 29).

12.4 Protocol for Macro characters in the first position

This protocol is used to encode two specific message headers and trailers in an abbreviated manner in Data Matrix symbols.

When a Macro character is in the first position a preamble and postamble shall be transmitted. If the first symbol character is 236 (i.e. encoding Macro 05), then the preamble [)>$^{R}{}_{S}$05$^{G}{}_{S}$ shall precede the encoded data that follows it. If the first symbol character is 237 (i.e. encoding Macro 06), then the preamble [)>$^{R}{}_{S}$06$^{G}{}_{S}$ shall precede the encoded data that follows it. The postamble $^{R}{}_{S}$$^{E}{}_{O_T}$ shall be transmitted after the data in both cases.

12.5 Protocol for ECIs

In systems where ECIs are supported, the use of a symbology identifier prefix is required with every transmission[1]. Whenever an ECI codeword is encountered, it shall be transmitted as the escape character 92_{DEC} (or $5C_{HEX}$), which represents the character "\" (backslash or reverse solidus) in the default interpretation. The next codeword(s) are converted into a 6-digit value, inverting the rules defined in Table 6. The 6-digit value is transmitted as the appropriate ASCII values (48 - 57). Application software recognising \nnnnnn should interpret all subsequent characters as being from the ECI defined by the 6-digit sequence. This interpretation remains in effect until the end of the encoded data or until another ECI sequence is encountered. If the backslash (byte 92_{DEC}) needs to be used as encoded data, transmission shall be as follows. Whenever (ASCII 92_{DEC}) occurs as data, two bytes of that value shall be transmitted, thus a single occurrence is always an escape character and a double occurrence indicates true data.

EXAMPLE

Encoded data: A\\B\C

Transmission: A\\\\B\\C

The use of the symbology identifier assures that the application can correctly interpret the escape character. An exception can be made if the ECI(s) can be handled entirely within the reader.

12.6 Symbology identifier

ISO/IEC 15424 provides a standard procedure for reporting the symbology which has been read, together with options set in the decoder and special features encountered in the symbol. Once the structure of the data (including the use of any ECI) has been identified, the appropriate symbology identifier should be added by the decoder as a preamble to the transmitted data. The symbology identifier is required if ECIs appear anywhere in the symbol (unless exception condition in 12.5 is valid), or if FNC1 is used as defined in 12.2 or 12.3. The symbology identifier and option values defined in Annex H shall be used.

12.7 Transmitted data example

In this example, the two-character message "¶Ж" is to be encoded, using the ASCII encodation scheme. "¶" is represented by a byte value of 182 in Data Matrix's default character set (ECI 000003, which is equivalent to ISO/IEC 8859-1). "Ж" is a Cyrillic character not available in ECI 000003, but which can be represented in ISO/IEC 8859-5 (see Reference [6]) (ECI 000007) by the same byte value of 182. The complete message can therefore be represented by inserting a switch to ECI 000007 after the first character, as follows: The symbol encodes the message <¶> <Switch to ECI 000007> <Ж>, using the following series of Data Matrix codewords: [Upper Shift] [55] [ECI] [8] [Upper Shift] [55], with decimal values of [235], [55], [241], [8], [235], [55].

NOTE 1 An Upper Shift character, followed by a codeword of value 55, encodes a byte value of 182.

NOTE 2 ECIs are encoded in Data Matrix as the ECI number plus one.

The decoder transmits the following bytes (including the symbology identifier prefix with an option value of 4, which indicates use of the ECI protocol):

93, 100, 52, 182, 92, 48, 48, 48, 48, 48, 55, 182

which, if viewed entirely in the default interpretation, would appear graphically as:]d4¶\000007¶

The decoder is responsible for signalling the switch to ECI 000007, but not for interpreting the result. ECI-aware software in the receiving application would delete the ECI escape sequence \000007, and the Cyrillic character "Ж" would be represented in a system-dependent manner (e.g. by changing the font in a desktop-publishing file). The final result would match the original message of "¶Ж".

Annex A
(normative)

Data Matrix interleaving process

A.1 Schematic illustration

Using the example of the 72 x 72 symbol size, four levels of interleaving are required to encode a total of 368 data codewords and 144 error correction codewords. These are divided into four blocks of 92 data codewords and 36 error correction codewords, a total block length of 128 codewords. This is depicted in Figure A.1.

	data codewords d	error correction codewords ε
Code-word stream	1 | 2 | 3 | 4 | … | … | 365 | 366 | 367 | 368	1 | 2 | 3 | 4 | … | … | 141 | 142 | 143 | 144
Block 1	1 | 5 | … | … | … | 361 | 365	1 | 5 | … | … | | 137 | 141
Block 2	2 | 6 | … | … | … | 362 | 366	2 | 6 | … | … | | 138 | 142
Block 3	3 | 7 | … | … | … | 363 | 367	3 | 7 | … | … | | 139 | 143
Block 4	4 | 8 | … | … | … | 364 | 368	4 | 8 | … | … | … | 140 | 144

Figure A.1 — Illustration of interleaving for 72 x 72 symbol

Symbol size of 144 x 144 is special in the sense that there are two block sizes, 156 for Block 1 to 8 and 155 for Block 9 and 10. 1558 data bytes and 620 ECC bytes are distributed in 10 blocks as depicted in Figure A.2.

Due to misunderstanding of a previous version of this document some implementations erroneously start ECC codewords from group 9 instead of group 1. Those implementations should follow this section.

Codeword stream	Data codewords d							Error correction codewords e					
	1	2	3	...	1556	1557	1558	1	2	3	...	619	620
Block 1	Data codewords d							Error correction codewords e					
	1	11	21	...	1541	1551		1	11	21	...	601	611
Block 2	Data codewords d							Error correction codewords e					
	2	12	22	...	1542	1552		2	12	22	...	602	612
Block 3	Data codewords d							Error correction codewords e					
	3	13	23	...	1543	1553		3	13	23	...	603	613
Block 4	Data codewords d							Error correction codewords e					
	4	14	24	...	1544	1554		4	14	24	...	604	614
Block 5	Data codewords d							Error correction codewords e					
	5	15	25	...	1545	1555		5	15	25	...	605	615
Block 6	Data codewords d							Error correction codewords e					
	6	16	26	...	1546	1556		6	16	26	...	606	616
Block 7	Data codewords d							Error correction codewords e					
	7	17	27	...	1547	1557		7	17	27	...	607	617
Block 8	Data codewords d							Error correction codewords e					
	8	18	28	...	1548	1558		8	18	28	...	608	618
Block 9	Data codewords d							Error correction codewords e					
	9	19	29	...	1549			9	19	29	...	609	619
Block 10	Data codewords d							Error correction codewords e					
	10	20	30	...	1550			10	20	30	...	610	620

Figure A.2 — Illustration of interleaving for 144 x 144 symbol

A.2 Starting sequence for interleaving in different sized symbols

The sequence of the interleaved data codewords and error correction codewords is given in Table A.1.

Table A.1 — Sequence of data and error correction codewords for different symbol sizes

Symbol size	Reed-Solomon block	Sequence of data codewords			Sequence of error correction codewords		
52 x 52	1	1, 3, 5	...	201, 203	1, 3, 5	...	81, 83
	2	2, 4, 6	...	202, 204	2, 4, 6	...	82, 84
64 x 64	1	1, 3, 5	...	277, 279	1, 3, 5	...	109, 111
	2	2, 4, 6	...	278, 280	2, 4, 6	...	110, 112
72 x 72	1	1, 5, 9	...	361, 365	1, 5, 9	...	137, 141
	2	2, 6, 10	...	362, 366	2, 6, 10	...	138, 142
	3	3, 7, 11	...	363, 367	3, 7, 11	...	139, 143
	4	4, 8, 12	...	364, 368	4, 8, 12	...	140, 144
80 x 80	1	1, 5, 9	...	449, 453	1, 5, 9	...	185, 189
	2	2, 6, 10	...	450, 454	2, 6, 10	...	186, 190
	3	3, 7, 11	...	451, 455	3, 7, 11	...	187, 191
	4	4, 8, 12	...	452, 456	4, 8, 12	...	188, 192
88 x 88	1	1, 5, 9	...	569, 573	1, 5, 9	...	217, 221
	2	2, 6, 10	...	570, 574	2, 6, 10	...	218, 222
	3	3, 7, 11	...	571, 575	3, 7, 11	...	219, 223
	4	4, 8, 12	...	572, 576	4, 8, 12	...	220, 224
96 x 96	1	1, 5, 9	...	689, 693	1, 5, 9	...	265, 269
	2	2, 6, 10	...	690, 694	2, 6, 10	...	266, 270
	3	3, 7, 11	...	691, 695	3, 7, 11	...	267, 271
	4	4, 8, 12	...	692, 696	4, 8, 12	...	268, 272
104 x 104	1	1, 7, 13	...	805, 811	1, 7, 13	...	325, 331
	2	2, 8, 14	...	806, 812	2, 8, 14	...	326, 332
	3	3, 9, 15	...	807, 813	3, 9, 15	...	327, 333
	4	4, 10, 16	...	808, 814	4, 10, 16	...	328, 334
	5	5, 11, 17	...	809, 815	5, 11, 17	...	329, 335
	6	6, 12, 18	...	810, 816	6, 12, 18	...	330, 336
120 x 120	1	1, 7, 13	...	1039, 1045	1, 7, 13	...	397, 403
	2	2, 8, 14	...	1040, 1046	2, 8, 14	...	398, 404
	3	3, 9, 15	...	1041, 1047	3, 9, 15	...	399, 405
	4	4, 10, 16	...	1042, 1048	4, 10, 16	...	400, 406
	5	5, 11, 17	...	1043, 1049	5, 11, 17	...	401, 407
	6	6, 12, 18	...	1044, 1050	6, 12, 18	...	402, 408
132 x 132	1	1, 9, 17	...	1289, 1297	1, 9, 17	...	481, 489
	2	2, 10, 18	...	1290, 1298	2, 10, 18	...	482, 490
	3	3, 11, 19	...	1291, 1299	3, 11, 19	...	483, 491
	4	4, 12, 20	...	1292, 1300	4, 12, 20	...	484, 492
	5	5, 13, 21	...	1293, 1301	5, 13, 21	...	485, 493
	6	6, 14, 22	...	1294, 1302	6, 14, 22	...	486, 494
	7	7, 15, 23	...	1295, 1303	7, 15, 23	...	487, 495
	8	8, 16, 24	...	1296, 1304	8, 16, 24	...	488, 496
144 x 144	1	1, 11, 21	...	1541, 1551	1, 11, 21	...	601, 611
	2	2, 12, 22	...	1542, 1552	2, 12, 22	...	602, 612
	3	3, 13, 23	...	1543, 1553	3, 13, 23	...	603, 613

Table A.1 *(continued)*

Symbol size	Reed-Solomon block	Sequence of data codewords			Sequence of error correction codewords		
	4	4, 14, 24	...	1544, 1554	4, 14, 24	...	604, 614
	5	5, 15, 25	...	1545, 1555	5, 15, 25	...	605, 615
	6	6, 16, 26	...	1546, 1556	6, 16, 26	...	606, 616
	7	7, 17, 27	...	1547, 1557	7, 17, 27	...	607, 617
	8	8, 18, 28	...	1548, 1558	8, 18, 28	...	608, 618
	9	9, 19, 29	...	1549	9, 19, 29	...	609, 619
	10	10, 20, 30	...	1550	10, 20, 30	...	610, 620

Annex B
(normative)

Data Matrix pattern randomising

B.1 General

The pattern randomising algorithms convert an input codeword at a given position to a new randomised output codeword.

B.2 253-state algorithm

B.2.1 General

This algorithm adds a pseudo-random number to the Pad codeword value. The pseudo-random number will always be in the range of 1 to 253 and the randomised Pad codeword value will be in the range 1 to 254.

The variable Pad_codeword_position is the number of data codewords from the beginning of encoded data.

B.2.2 253-state randomising algorithm

INPUT (Pad_codeword_value, Pad_codeword_position)

pseudo_random_number = ((149 * Pad_codeword_position) mod 253) + 1

temp_variable = Pad_codeword_value + pseudo_random_number

IF (temp_variable ≤ 254)

  OUTPUT (randomised_Pad_codeword_value = temp_variable)

ELSE

  OUTPUT (randomised_Pad_codeword_value = temp_variable − 254)

B.2.3 253-state un-randomising algorithm

INPUT (randomised_Pad_codeword_value, Pad_codeword_position)

pseudo_random_number = ((149 * Pad_codeword_position) mod 253) + 1

temp_variable = randomised_Pad_codeword_value − pseudo_random_number

IF (temp_variable ≥ 1)

  OUTPUT (Pad_codeword_value = temp_variable)

ELSE

  OUTPUT (Pad_codeword_value = temp_variable + 254)

B.3 255-state algorithm

B.3.1 General

This algorithm adds a pseudo-random number to the Base 256 encodation codeword value. The pseudorandom number will always be in the range of 1 to 255 and the randomised Base 256 codeword value will be in the range 0 to 255.

The variable Base256_codeword_position is the number of data codewords from the beginning of encoded data.

B.3.2 255-state randomising algorithm

INPUT (Base256_codeword_value, Base256_codeword_position)

pseudo_random_number = ((149 * Base256_codeword_position) mod 255) + 1

temp_variable = Base256_codeword_value + pseudo_random_number

IF (temp_variable ≤ 255)

OUTPUT (randomised_Base256_codeword_value = temp_variable)

ELSE

OUTPUT (randomised_Base256_codeword_value = temp_variable – 256)

B.3.3 255-state un-randomising algorithm

INPUT (randomised_Base256_codeword_value, Base256_codeword_position)

pseudo_random_number = ((149 * Base256_codeword_position) mod 255) + 1

temp_variable=randomised_Base256_codeword_value – pseudo_random_number

IF (temp_variable ≥ 0)

OUTPUT (Base256_codeword_value = temp_variable)

ELSE

OUTPUT (Base256_codeword_value = temp_variable + 256)

Annex C
(normative)

Data Matrix encodation character sets

Table C.1 — C40 encodation character set

C40 Value	Basic set		Shift 1 set		Shift 2 set		Shift 3 set	
	Char	Decimal	Char	Decimal	Char	Decimal	Char	Decimal
0	Shift 1		NUL	0	!	33	‘	96
1	Shift 2		SOH	1	“	34	a	97
2	Shift 3		STX	2	#	35	b	98
3	space	32	ETX	3	$	36	c	99
4	0	48	EOT	4	%	37	d	100
5	1	49	ENQ	5	&	38	e	101
6	2	50	ACK	6	‘	39	f	102
7	3	51	BEL	7	(	40	g	103
8	4	52	BS	8	)	41	h	104
9	5	53	HT	9	*	42	i	105
10	6	54	LF	10	+	43	j	106
11	7	55	VT	11	,	44	k	107
12	8	56	FF	12	-	45	l	108
13	9	57	CR	13	.	46	m	109
14	A	65	SO	14	/	47	n	110
15	B	66	SI	15	:	58	o	111
16	C	67	DLE	16	;	59	p	112
17	D	68	DC1	17	<	60	q	113
18	E	69	DC2	18	=	61	r	114
19	F	70	DC3	19	>	62	s	115
20	G	71	DC4	20	?	63	t	116
21	H	72	NAK	21	@	64	u	117
22	I	73	SYN	22	[	91	v	118
23	J	74	ETB	23	\	92	w	119
24	K	75	CAN	24	]	93	x	120
25	L	76	EM	25	^	94	y	121
26	M	77	SUB	26	_	95	z	122
27	N	78	ESC	27	FNC1		{	123
28	O	79	FS	28			|	124
29	P	80	GS	29			}	125
30	Q	81	RS	30	Upper Shift		~	126
31	R	82	US	31			DEL	127
32	S	83						
33	T	84						
34	U	85						
35	V	86						
36	W	87						

Table C.1 *(continued)*

C40 Value	Basic set		Shift 1 set		Shift 2 set		Shift 3 set	
	Char	Decimal	Char	Decimal	Char	Decimal	Char	Decimal
37	X	88						
38	Y	89						
39	Z	90						

NOTE The relationship between the ASCII decimal value and the C40 value remains constant regardless of which ECI is in effect.

Table C.2 — Text encodation character set

Text value	Basic set		Shift 1 set		Shift 2 set		Shift 3 set	
	Char	Decimal	Char	Decimal	Char	Decimal	Char	Decimal
0	Shift	1	NUL	0	!	33	‘	96
1	Shift	2	SOH	1	“	34	A	65
2	Shift	3	STX	2	#	35	B	66
3	space	32	ETX	3	$	36	C	67
4	0	48	EOT	4	%	37	D	68
5	1	49	ENQ	5	&	38	E	69
6	2	50	ACK	6	‘	39	F	70
7	3	51	BEL	7	(	40	G	71
8	4	52	BS	8	)	41	H	72
9	5	53	HT	9	*	42	I	73
10	6	54	LF	10	+	43	J	74
11	7	55	VT	11	,	44	K	75
12	8	56	FF	12	-	45	L	76
13	9	57	CR	13	.	46	M	77
14	a	97	SO	14	/	47	N	78
15	b	98	SI	15	:	58	O	79
16	c	99	DLE	16	;	59	P	80
17	d	100	DC1	17	<	60	Q	81
18	e	101	DC2	18	=	61	R	82
19	f	102	DC3	19	>	62	S	83
20	g	103	DC4	20	?	63	T	84
21	h	104	NAK	21	@	64	U	85
22	i	105	SYN	22	[	91	V	86
23	j	106	ETB	23	\	92	W	87
24	k	107	CAN	24	]	93	X	88
25	l	108	EM	25	^	94	Y	89
26	m	109	SUB	26	_	95	Z	90
27	n	110	ESC	27	FNC1		{	123
28	o	111	FS	28			|	124
29	p	112	GS	29			}	125
30	q	113	RS	30	Upper	Shift	~	126
31	r	114	US	31			DEL	127
32	s	115						
33	t	116						
34	u	117						
35	v	118						

Table C.2 *(continued)*

Text value	Basic set		Shift 1 set		Shift 2 set		Shift 3 set	
	Char	Decimal	Char	Decimal	Char	Decimal	Char	Decimal
36	w	119						
37	x	120						
38	y	121						
39	z	122						

NOTE The relationship between the ASCII decimal value and the Text value remains constant regardless of which ECI is in effect.

Table C.3 — EDIFACT encodation character set

Data character			EDIFACT binary value	Data character			EDIFACT binary value
Char	Decimal value	Binary value		Char	Decimal value	Binary value	
@	64	01000000	000000	space	32	00100000	100000
A	65	01000001	000001	!	33	00100001	100001
B	66	01000010	000010	“	34	00100010	100010
C	67	01000011	000011	#	35	00100011	100011
D	68	01000100	000100	$	36	00100100	100100
E	69	01000101	000101	%	37	00100101	100101
F	70	01000110	000110	&	38	00100110	100110
G	71	01000111	000111	‘	39	00100111	100111
H	72	01001000	001000	(	40	00101000	101000
I	73	01001001	001001	)	41	00101001	101001
J	74	01001010	001010	*	42	00101010	101010
K	75	01001011	001011	+	43	00101011	101011
L	76	01001100	001100	,	44	00101100	101100
M	77	01001101	001101	-	45	00101101	101101
N	78	01001110	001110	.	46	00101110	101110
O	79	01001111	001111	/	47	00101111	101111
P	80	01010000	010000	0	48	00110000	110000
Q	81	01010001	010001	1	49	00110001	110001
R	82	01010010	010010	2	50	00110010	110010
S	83	01010011	010011	3	51	00110011	110011
T	84	01010100	010100	4	52	00110100	110100
U	85	01010101	010101	5	53	00110101	110101
V	86	01010110	010110	6	54	00110110	110110
W	87	01010111	010111	7	55	00110111	110111
X	88	01011000	011000	8	56	00111000	111000
Y	89	01011001	011001	9	57	00111001	111001
Z	90	01011010	011010	:	58	00111010	111010
[	91	01011011	011011	;	59	00111011	111011
\	92	01011100	011100	<	60	00111100	111100
]	93	01011101	011101	=	61	00111101	111101
^	94	01011110	011110	>	62	00111110	111110
Unlatch		01011111	011111	?	63	00111111	111111

NOTE The relationship between the ASCII decimal value and the EDIFACT value remain constant regardless of which ECI is in effect.

Annex D
(normative)

Data Matrix alignment patterns

Figures D.1 to D.4 are examples of alignment patterns.

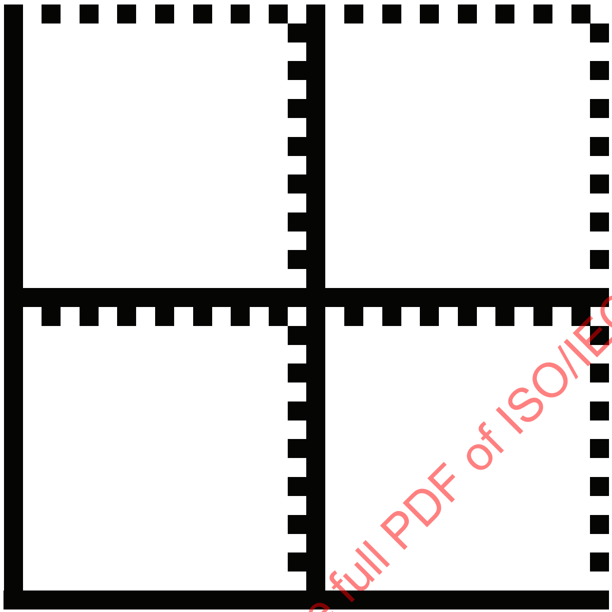


Figure D.1 — Alignment pattern configuration for 32 x 32 square symbol

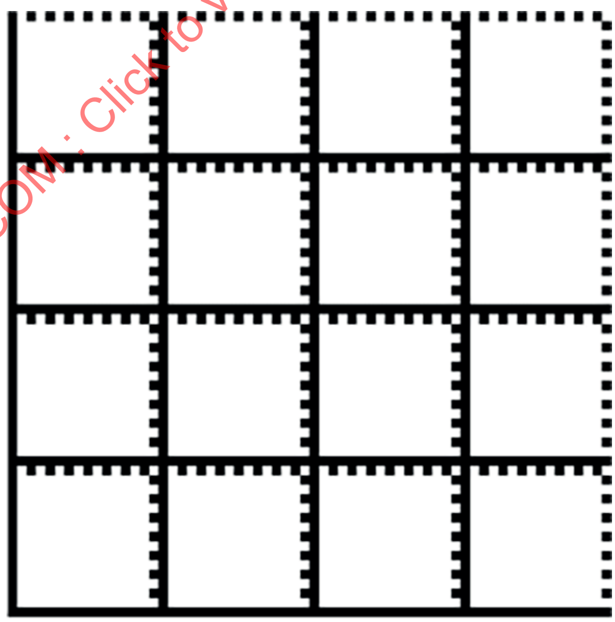


Figure D.2 — Alignment pattern configuration for 64 x 64 square symbol

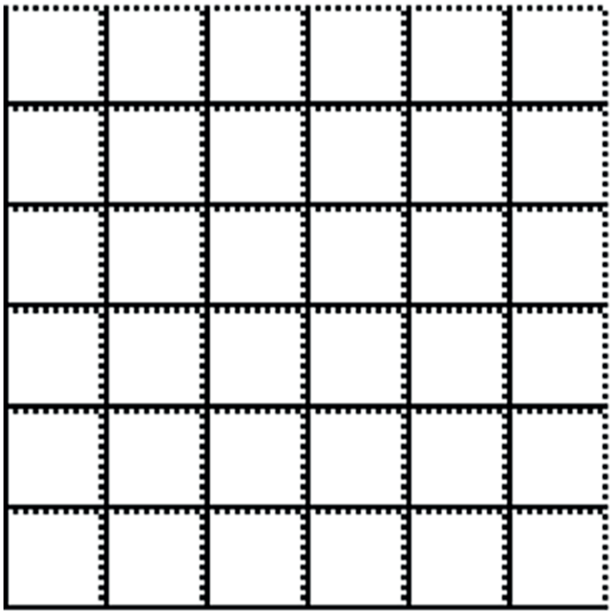


Figure D.3 — Alignment pattern configuration for 120 x 120 square symbol

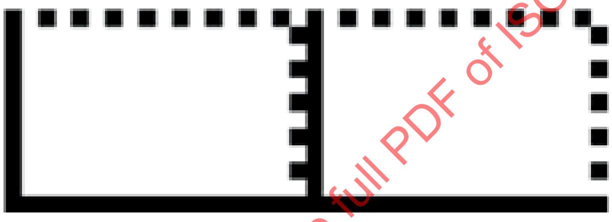


Figure D.4 — Alignment pattern configuration for 12 x 36 rectangular symbol

Annex E
(normative)

Data Matrix Reed-Solomon error detection and correction

E.1 Error correction codeword generator polynomials

The error correction codewords are the coefficients of the remainder resulting from first multiplying the symbol data polynomial d(x) by x^k and then dividing it by the generator polynomial $g(x)$. Each generator polynomial is the product of the first-degree polynomials: $x - 2^1$, $x - 2^2$, ..., $x - 2^n$; where n is the degree of the generator polynomial.

For example, the fifth degree generator polynomial is:

$(x + 2)(x + 4)(x + 8)(x + 16)(x + 32)$

$= x^5 + (2 + 4 + 8 + 16 + 32)x^4 + ((2 * 4) + (2 * 8) + (2 * 16) + (2 * 32) + (4 * 8) + (4 * 16) + (4 * 32) + (8 * 16) + (8 * 32) + (16 * 32))x^3 + ((2 * 4 * 8) + (2 * 4 * 16) + (2 * 4 * 32) + (2 * 8 * 16) + (2 * 8 * 32) + (2 * 16 * 32) + (4 * 8 * 16) + (4 * 8 * 32) + (4 * 16 * 32) + (8 * 16 * 32)) x^2 + ((2 * 4 * 8 * 16) + (2 * 4 * 8 * 32) + (2 * 4 * 16 * 32) + (2 * 8 * 16 * 32) + (4 * 8 * 16 * 32))x + (2 * 4 * 8 * 16 * 32)$

$= x^5 + 62x^4 + 111x^3 + 15x^2 + 48x + 228.$

Note that this Galois Field arithmetic is not normal integer arithmetic: − is equivalent to +, which is an "exclusive-or" operation in this Field, and multiplication is byte-wise modulo 100101101 for each binary polynomial term generated by bit-by-bit multiplication.

The polynomial divisor for generating 5 check characters is:

$g(x) = x^5 + 62x^4 + 111x^3 + 15x^2 + 48x + 228.$

The polynomial divisor for generating 7 check characters is:

$g(x) = x^7 + 254x^6 + 92x^5 + 240x^4 + 134x^3 + 144x^2 + 68x + 23.$

The polynomial divisor for generating 10 check characters is:

$g(x) = x^{10} + 61x^9 + 110x^8 + 255x^7 + 116x^6 + 248x^5 + 223x^4 + 166x^3 + 185x^2 + 24x + 28.$

The polynomial divisor for generating 11 check characters is:

$g(x) = x^{11} + 120x^{10} + 97x^9 + 60x^8 + 245x^7 + 39x^6 + 168x^5 + 194x^4 + 12x^3 + 205x^2 + 138x + 175.$

The polynomial divisor for generating 12 check characters is:

$g(x) = x^{12} + 242x^{11} + 100x^{10} + 178x^9 + 97x^8 + 213x^7 + 142x^6 + 42x^5 + 61x^4 + 91x^3 + 158x^2 + 153x + 41.$

The polynomial divisor for generating 14 check characters is:

$g(x) = x^{14} + 185x^{13} + 83x^{12} + 186x^{11} + 18x^{10} + 45x^9 + 138x^8 + 119x^7 + 157x^6 + 9x^5 + 95x^4 + 252x^3 + 192x^2 + 97x + 156.$

The polynomial divisor for generating 18 check characters is:

$g(x) = x^{18} + 188x^{17} + 90x^{16} + 48x^{15} + 225x^{14} + 254x^{13} + 94x^{12} + 129x^{11} + 109x^{10} + 213x^9 + 241x^8 + 61x^7 + 66x^6 + 75x^5 + 188x^4 + 39x^3 + 100x^2 + 195x + 83.$

The polynomial divisor for generating 20 check characters is:

$g(x) = x^{20} + 172x^{19} + 186x^{18} + 174x^{17} + 27x^{16} + 82x^{15} + 108x^{14} + 79x^{13} + 253x^{12} + 145x^{11} + 153x^{10} + 160x^{9} + 188x^{8} + 2x^{7} + 168x^{6} + 71x^{5} + 233x^{4} + 9x^{3} + 244x^{2} + 195x + 15.$

The polynomial divisor for generating 24 check characters is:

$g(x) = x^{24} + 193x^{23} + 50x^{22} + 96x^{21} + 184x^{20} + 181x^{19} + 12x^{18} + 124x^{17} + 254x^{16} + 172x^{15} + 5x^{14} + 21x^{13} + 155x^{12} + 223x^{11} + 251x^{10} + 197x^{9} + 155x^{8} + 21x^{7} + 176x^{6} + 39x^{5} + 109x^{4} + 205x^{3} + 88x^{2} + 190x + 52.$

The polynomial divisor for generating 28 check characters is:

$g(x) = x^{28} + 255x^{27} + 93x^{26} + 168x^{25} + 233x^{24} + 151x^{23} + 120x^{22} + 136x^{21} + 141x^{20} + 213x^{19} + 110x^{18} + 138x^{17} + 17x^{16} + 121x^{15} + 249x^{14} + 34x^{13} + 75x^{12} + 53x^{11} + 170x^{10} + 151x^{9} + 37x^{8} + 174x^{7} + 103x^{6} + 96x^{5} + 71x^{4} + 97x^{3} + 43x^{2} + 231x + 211.$

The polynomial divisor for generating 36 check characters is:

$g(x) = x^{36} + 112x^{35} + 81x^{34} + 98x^{33} + 225x^{32} + 25x^{31} + 59x^{30} + 184x^{29} + 175x^{28} + 44x^{27} + 115x^{26} + 119x^{25} + 95x^{24} + 137x^{23} + 101x^{22} + 33x^{21} + 68x^{20} + 4x^{19} + 2x^{18} + 18x^{17} + 229x^{16} + 182x^{15} + 80x^{14} + 251x^{13} + 220x^{12} + 179x^{11} + 84x^{10} + 120x^{9} + 102x^{8} + 181x^{7} + 162x^{6} + 250x^{5} + 130x^{4} + 218x^{3} + 242x^{2} + 127x + 245.$

The polynomial divisor for generating 42 check characters is:

$g(x) = x^{42} + 5x^{41} + 9x^{40} + 5x^{39} + 226x^{38} + 177x^{37} + 150x^{36} + 50x^{35} + 69x^{34} + 202x^{33} + 248x^{32} + 101x^{31} + 54x^{30} + 57x^{29} + 253x^{28} + x^{27} + 21x^{26} + 121x^{25} + 57x^{24} + 111x^{23} + 214x^{22} + 105x^{21} + 167x^{20} + 9x^{19} + 100x^{18} + 95x^{17} + 175x^{16} + 8x^{15} + 242x^{14} + 133x^{13} + 245x^{12} + 2x^{11} + 122x^{10} + 105x^{9} + 247x^{8} + 153x^{7} + 22x^{6} + 38x^{5} + 19x^{4} + 31x^{3} + 137x^{2} + 193x + 77.$

The polynomial divisor for generating 48 check characters is:

$g(x) = x^{48} + 19x^{47} + 225x^{46} + 253x^{45} + 92x^{44} + 213x^{43} + 69x^{42} + 175x^{41} + 160x^{40} + 147x^{39} + 187x^{38} + 87x^{37} + 176x^{36} + 44x^{35} + 82x^{34} + 240x^{33} + 186x^{32} + 138x^{31} + 66x^{30} + 100x^{29} + 120x^{28} + 88x^{27} + 131x^{26} + 205x^{25} + 170x^{24} + 90x^{23} + 37x^{22} + 23x^{21} + 118x^{20} + 147x^{19} + 16x^{18} + 106x^{17} + 191x^{16} + 87x^{15} + 237x^{14} + 188x^{13} + 205x^{12} + 231x^{11} + 238x^{10} + 133x^{9} + 238x^{8} + 22x^{7} + 117x^{6} + 32x^{5} + 96x^{4} + 223x^{3} + 172x^{2} + 132x + 245.$

The polynomial divisor for generating 56 check characters is:

$g(x) = x^{56} + 46x^{55} + 143x^{54} + 53x^{53} + 233x^{52} + 107x^{51} + 203x^{50} + 43x^{49} + 155x^{48} + 28x^{47} + 247x^{46} + 67x^{45} + 127x^{44} + 245x^{43} + 137x^{42} + 13x^{41} + 164x^{40} + 207x^{39} + 62x^{38} + 117x^{37} + 201x^{36} + 150x^{35} + 22x^{34} + 238x^{33} + 144x^{32} + 232x^{31} + 29x^{30} + 203x^{29} + 117x^{28} + 234x^{27} + 218x^{26} + 146x^{25} + 228x^{24} + 54x^{23} + 132x^{22} + 200x^{21} + 38x^{20} + 223x^{19} + 36x^{18} + 159x^{17} + 150x^{16} + 235x^{15} + 215x^{14} + 192x^{13} + 230x^{12} + 170x^{11} + 175x^{10} + 29x^{9} + 100x^{8} + 208x^{7} + 220x^{6} + 17x^{5} + 12x^{4} + 238x^{3} + 223x^{2} + 9x + 175.$

The polynomial divisor for generating 62 check characters is:

$g(x) = x^{62} + 204x^{61} + 11x^{60} + 47x^{59} + 86x^{58} + 124x^{57} + 224x^{56} + 166x^{55} + 94x^{54} + 7x^{53} + 232x^{52} + 107x^{51} + 4x^{50} + 170x^{49} + 176x^{48} + 31x^{47} + 163x^{46} + 17x^{45} + 188x^{44} + 130x^{43} + 40x^{42} + 10x^{41} + 87x^{40} + 63x^{39} + 51x^{38} + 218x^{37} + 27x^{36} + 6x^{35} + 147x^{34} + 44x^{33} + 161x^{32} + 71x^{31} + 114x^{30} + 64x^{29} + 175x^{28} + 221x^{27} + 185x^{26} + 106x^{25} + 250x^{24} + 190x^{23} + 197x^{22} + 63x^{21} + 245x^{20} + 230x^{19} + 134x^{18} + 112x^{17} + 185x^{16} + 37x^{15} + 196x^{14} + 108x^{13} + 143x^{12} + 189x^{11} + 201x^{10} + 188x^{9} + 202x^{8} + 118x^{7} + 39x^{6} + 210x^{5} + 144x^{4} + 50x^{3} + 169x^{2} + 93x + 242.$

The polynomial divisor for generating 68 check characters is:

$g(x) = x^{68} + 186x^{67} + 82x^{66} + 103x^{65} + 96x^{64} + 63x^{63} + 132x^{62} + 153x^{61} + 108x^{60} + 54x^{59} + 64x^{58} + 189x^{57} + 211x^{56} + 232x^{55} + 49x^{54} + 25x^{53} + 172x^{52} + 52x^{51} + 59x^{50} + 241x^{49} + 181x^{48} + 239x^{47} + 223x^{46} + 136x^{45} + 231x^{44} + 210x^{43} + 96x^{42} + 232x^{41} + 220x^{40} + 25x^{39} + 179x^{38} + 167x^{37} + 202x^{36} + 185x^{35} + 153x^{34} + 139x^{33} + 66x^{32} + 236x^{31} + 227x^{30} + 160x^{29} + 15x^{28} + 213x^{27} + 93x^{26} + 122x^{25} + 68x^{24} + 177x^{23} + 158x^{22} + 197x^{21} + 234x^{20} + 180x^{19} + 248x^{18} + 136x^{17} + 213x^{16} + 127x^{15} + 73x^{14} + 36x^{13} + 154x^{12} + 244x^{11} + 147x^{10} + 33x^{9} + 89x^{8} + 56x^{7} + 159x^{6} + 149x^{5} + 251x^{4} + 89x^{3} + 173x^{2} + 228x + 220.$

E.2 Error correction calculation

The Peterson-Gorenstein-Zierler algorithm may be used to correct errors in decoded Data Matrix symbols.

The calculation described below follows this error correcting algorithm, using the Reed-Solomon error correction codewords.

Erasures shall be corrected as errors by initially filling any erasure codeword positions with dummy values.

All calculations shall be done using GF(2^8) arithmetic operations. Addition and subtraction are equivalent to the binary XOR operation. Multiplication and division can be performed using log and antilog tables.

Construct the symbol character polynomial $C(x) = C_{n-1}x^{n-1} + C_{n-2}x^{n-2} + ... + C_1x^1 + C_0$ where the n coefficients are the codewords read with C_{n-1} being the first symbol character and where n is the total number of symbol characters.

Calculate i syndrome values S_0 through S_{i-1} by evaluating $C(x)$ at $x = 2^k$ for $k = 1$ through i, where i is the number of error correction codewords in the symbol.

Form and solve j simultaneous equations with j unknowns L_0 through L_{j-1} using the i syndromes:

$$S_0L_0 + S_1L_1 + ... + S_{j-1}L_{j-1} = S_j$$

$$S_1L_0 + S_2L_1 + ... + S_jL_{j-1} = S_{j+1}$$

$$\vdots$$

$$\vdots$$

$$S_{j-1}L_0 + S_jL_1 + ... + S_{2j-2}L_{j-1} = S_{2j-1}$$

where $j = i/2$.

Construct the error locator polynomial:

$$L(x) = L_{j-1}x^j + L_{j-2}x^{j-1} + ... + L_0x + 1$$

from the j values of L obtained above. Evaluate $L(x)$ at $x = 2^k$ for $k = 0$ through $n - 1$ where n is the total number of symbol characters in the symbol.

Whenever $L(2^k) = 0$, an error location is given by $n - 1 - k$. If more than j error locations are found, the symbol is not correctable.

Save the error locations in m error location variables E_0 through E_{m-1} where m is the number of error locations found. Form and solve m simultaneous equations with m unknowns X_0 through X_{m-1} (the error magnitudes) using the error location variables E and the first m syndromes S:

$$E_0X_0 + E_1X_1 + ... + E_{m-1}X_{m-1} = S_0$$

$$E_0^2X_0 + E_1^2X_1 + ... + E_{(m-1)}^2X_{m-1} = S_1$$

$$E_0^3X_0 + E_1^3X_1 + ... + E_{(m-1)}^3X_{m-1} = S_2$$

$$\vdots$$

$$\vdots$$

$$E_0^mX_0 + E_1^mX_1 + ... + E_{(m-1)}^mX_{m-1} = S_{m-1}$$

Add the error magnitudes X_0 through X_{m-1} to the symbol character values at the corresponding error locations E_0 through E_{m-1} to correct the errors.

NOTE $E_0 ... E_{m-1}$ – are the roots of the error locator polynomial.

E.3 Calculation of error correction codewords

The following is an example of a generic routine, written in C, which calculates the error correction codewords for a given data codeword string of length "nd", stored as an integer array wd[]. The function ReedSolomon() first generates log and antilog tables for the Galois Field of size "gf" (= 2^8) with prime modulus "pp" (= 301), then uses them in the function prod(), first to calculate coefficients of the generator polynomial of order "nc" and then to calculate "nc" additional check codewords which are appended to the data in wd[5],[6].

```
/* "prod(x,y,log,alog,gf)" returns the product "x" times "y" */
int prod(int x, int y, int *log, int *alog, int gf) {
   if (!x || !y) return 0;
   ELSE return alog[(log[x] + log[y]) % (gf-1)];
}

/* "ReedSolomon(wd,nd,nc,gf.pp)" takes "nd" data codeword values in wd[] */
/* and adds on "nc" check codewords, all within GF(gf) where "gf" is a */
/* power of 2 and "pp" is the value of its prime modulus polynomial */
void ReedSolomon(int *wd, int nd, int nc, int gf, int pp) {
   int i, j, k. *log,*alog,*c;

/* allocate, then generate the log & antilog arrays: */
   log = malloc(sizeof(int) * gf);
   alog = malloc(sizeof(int) * gf);
   log[0] = 1-gf; alog[0] = 1;
   for (i = 1; i < gf; i++) {
      alog[i] = alog[i-1] * 2;
      if (alog[i] >= gf) alog[i] ^= pp;
      log[alog[i]] = i;
}

/* allocate, then generate the generator polynomial coefficients: */
   c = malloc(sizeof(int) * (nc+1));
   for (i=1; i<=nc; i++) c[i] = 0; c[0] = 1;
   for (i=1; i<=nc; i++) {
      c[i] = c[i-1];
      for (j=i-1; j>=1; j--) {
         c[j] = c[j-1] ^ prod(c[j],alog[i],log,alog,gf);
      }
      c[0] = prod(c[0],alog[i],log,alog,gf);
   }

/* clear, then generate "nc" checkwords in the array wd[] : */
   for (i=nd; i<=(nd+nc); i++) wd[i] = 0;
   for (i=0; i<nd; i++) {
      k = wd[nd] ^ wd[i] ;
      for (j=0; j<nc; j++) {
         wd[nd+j] = wd[nd+j+1] ^ prod(k,c[nc-j-1],log, alog,gf);
      }
   }

   free(c);
   free(alog);
   free(log);
}
```

Annex F
(normative)

Symbol character placement

F.1 Symbol character placement sample program

The following C language program generates symbol character placement diagrams:

```
#include <stdio.h>
#include <alloc.h>

int nrow, ncol, *array;

/* "module" places "chr+bit" with appropriate wrapping within array[] */
void module(int row, int col, int chr, int bit)
{ if (row < 0) { row += nrow; col += 4 - ((nrow+4)%8); }
   if (col < 0) { col += ncol; row += 4 - ((ncol+4)%8); }
   array[row*ncol+col] = 10*chr + bit;
}
/* "utah" places the 8 bits of a utah-shaped symbol character */
void utah(int row, int col, int chr)
{ module(row-2,col-2,chr,1);
   module(row-2,col-1,chr,2);
   module(row-1,col-2,chr,3);
   module(row-1,col-1,chr,4);
   module(row-1,col,chr,5);
   module(row,col-2,chr,6);
   module(row,col-1,chr,7);
   module(row,col,chr,8);
}
/* "cornerN" places 8 bits of the four special corner cases */
void corner1(int chr)
{ module(nrow-1,0,chr,1);
   module(nrow-1,1,chr,2);
   module(nrow-1,2,chr,3);
   module(0,ncol-2,chr,4);
   module(0,ncol-1,chr,5);
   module(1,ncol-1,chr,6);
   module(2,ncol-1,chr,7);
   module(3,ncol-1,chr,8);
}
void corner2(int chr)
{ module(nrow-3,0,chr,1);
   module(nrow-2,0,chr,2);
   module(nrow-1,0,chr,3);
   module(0,ncol-4,chr,4);
   module(0,ncol-3,chr,5);
   module(0,ncol-2,chr,6);
   module(0,ncol-1,chr,7);
   module(1,ncol-1,chr,8);
}
void corner3(int chr)
{ module(nrow-3,0,chr,1);
   module(nrow-2,0,chr,2);
   module(nrow-1,0,chr,3);
   module(0,ncol-2,chr,4);
   module(0,ncol-1,chr,5);
   module(1,ncol-1,chr,6);
   module(2,ncol-1,chr,7);
   module(3,ncol-1,chr,8);
}
void corner4(int chr)
{ module(nrow-1,0,chr,1);
```

```
   module(nrow-1,ncol-1,chr,2);
   module(0,ncol-3,chr,3);
   module(0,ncol-2,chr,4);
   module(0,ncol-1,chr,5);
   module(1,ncol-3,chr,6);
   module(1,ncol-2,chr,7);
   module(1,ncol-1,chr,8);
}
/* "PlaceDM" fills an nrow x ncol array with appropriate values */
void PlaceDM (void)
{ int row, col, chr;

/* First, fill the array[] with invalid entries */
   for (row=0; row<nrow; row++) {
      for (col=0; col<ncol; col++) {
         array[row*ncol+col] = 0;
      }
   }
/* Starting in the correct location for character #1, bit 8,... */
   chr = 1; row = 4; col = 0;

   do {
/* repeatedly first check for one of the special corner cases, then... */
      if ((row == nrow) && (col == 0)) corner1(chr++);
      if ((row == nrow-2) && (col == 0) && (ncol%4)) corner2(chr++);
      if ((row == nrow-2) && (col == 0) && (ncol%8 == 4)) corner3(chr++);
      if ((row == nrow+4) && (col == 2) && (!(ncol%8))) corner4(chr++);
/* sweep upward diagonally, inserting successive characters,... */
   do {
      if ((row < nrow) && (col >= 0) && (!array[row*ncol+col]))
         utah(row,col,chr++);
      row -= 2; col += 2;
   } while ((row >= 0) && (col < ncol));
   row += 1; col += 3;

/* & then sweep downward diagonally, inserting successive characters,... */
   +
   do {
      if ((row >= 0) && (col < ncol) && (!array[row*ncol+col]))
         utah(row,col,chr++);
      row += 2; col -= 2;
   } while ((row < nrow) && (col >= 0));
   row += 3; col += 1;

/* ... until the entire array is scanned */
   } while ((row < nrow) || (col < ncol));

/* Lastly, if the lower righthand corner is untouched, fill in fixed pattern */
   if (!array[nrow*ncol-1]) {
      array[nrow*ncol-1] = array[nrow*ncol-ncol-2] = 1;
   }
}

/* "main" checks for valid command line entries, then computes & displays array */
void main(int argc, char *argv[])
{ int x, y, z;

   if (argc =< 3) {
      printf("Command line: #_of_Data_Rows #_of_Data_Columns\n");
   } ELSE {
      nrow = ncol = 0;
      nrow = atoi(argv[1]); ncol = atoi(argv[2]);
      if ((nrow >= 6) && (~nrow&0x01) && (ncol >= 6) && (~ncol&0x01)) {
         array = malloc(sizeof(int) * nrow * ncol);

         PlaceDM();

         for (x=0; x<nrow; x++) {
            for (y=0; y<ncol; y++) {
               z = array[x*ncol+y];
               if (z == 0) printf(" WHI");
                  ELSE if (z == 1) printf("BLK");
```

```
                    ELSE printf("%3d.%d",z/10,z%10);
                }
                printf("\n");
            }
            free(array);
        }
    }
}
```

F.2 Symbol character placement rules

F.2.1 Non-standard symbol character shapes

Because the standard symbol character shape cannot always fit at the data module boundaries of the symbol and at some corners, a small set of non-standard symbol characters is required. There are six conditions: two boundary conditions which affect all symbol formats, and four different corner conditions which apply to certain symbol formats.

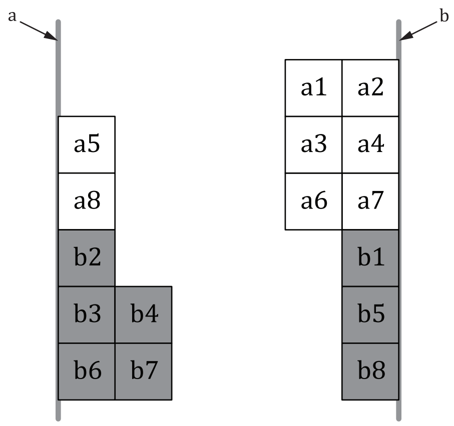
a) One portion of the symbol character shape is placed on one side and the other on the opposite side. This applies to two basic symbol character shapes (see Figure F.1). Variants of these arrangements concern the row-to-row relationship between the left- and right-hand boundary (see Table F.1).

b) One portion of the symbol character is placed on the top boundary and the other portion on the bottom boundary. This applies to two basic symbol character shapes (see Figure F.2). Variants of these arrangements concern the column-to-column relationship between the top and bottom boundary (see Table F.1).

c) Four symbol character shapes are split between two or three corners (see Figures F.3 to F.6). The non-standard symbol shapes are placed at opposite boundaries. The number of these pairings increases in general proportion to the size of the perimeter of the mapping matrix. The basic pattern is as illustrated in Figures F.1 and F.2. In Figure F.1, modules a8 and a7 are in the same row, as are modules b7 and b6. In Figure F.2, module c6 and c3 are in the same column as are modules d3 and d1. There are seven cases for boundary placement, which define the relative vertical position of the symbol characters illustrated in Figure F.1, the horizontal position of the symbol characters illustrated in Figure F.2, and the corner conditions.

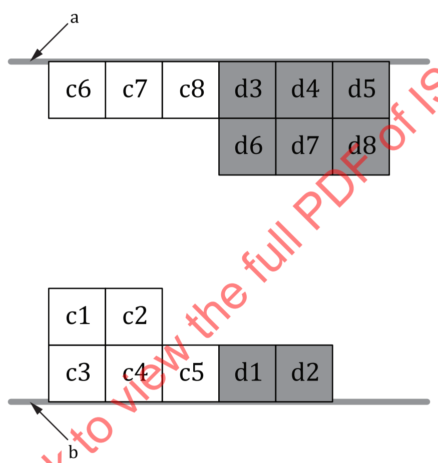
Table F.1 — Factors which determine the boundary placement cases

Boundary placement case	Row relationship of module a8 and a7	Column relationship of module c6 and c3	Corner condition Figure no.	Mapping matrices affected	Refer to Annex F. Figure no. for example
1	a7 Row = a8 Row	c3 Column = c6 Column	None	Square: 8^2, 16^2, 24^2, 32^2, 40^2 48^2, 56^2, 64^2, 72^2, 80^2, 88^2, 96^2, 120^2	Figure F.9, F.16
2	a7 Row = a8 Row − 2	c3 Column = c6 Column − 2	None	Square: 10^2, 18^2	Figure F.10, F.17
3	a7 Row = a8 Row + 4	c3 Column = c6 Column + 4	F.3	Square: 12^2, 20^2,28^2, 36^2, 44^2, 108^2, 132^2	Figure F.11, F.18
4	a7 Row = a8 Row + 2	c3 Column = c6 Column + 2	F.4	Square: 14^2, 22^2	Figure F.12, F.19
5	a7 Row = a8 Row	c3 Column = c6 Column + 2	F.5	Rectangular: 6 x 16, 14 x 32	Figure F.13
6	a7 Row = a8 Row	c3 Column = c6 Column − 2	None	Rectangular: 10 x 24, 10 x 32	Figure F.14
7	a7 Row = a8 Row + 4	c3 Column = c6 Column + 2	F.6	Rectangular: 6 x 28, 14 x 44	Figure F.15



a Left boundary.
b Right boundary.

Figure F.1 — Left and right symbol characters



a Top boundary.
b Bottom boundary.

Figure F.2 — Top and bottom symbol characters

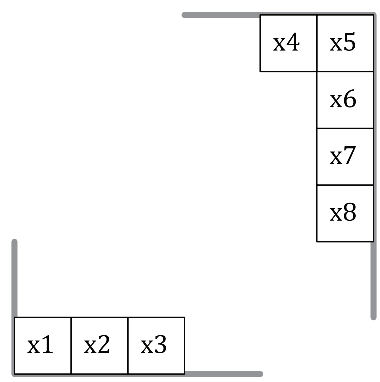


Figure F.3 — Corner condition 1

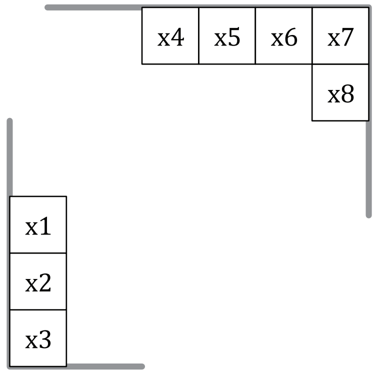


Figure F.4 — Corner condition 2

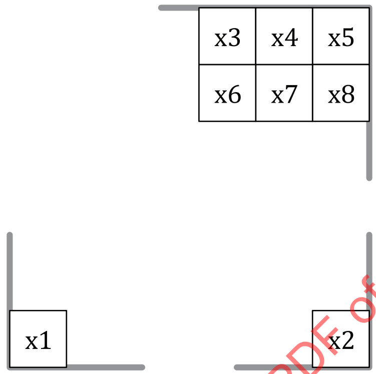


Figure F.5 — Corner condition 3

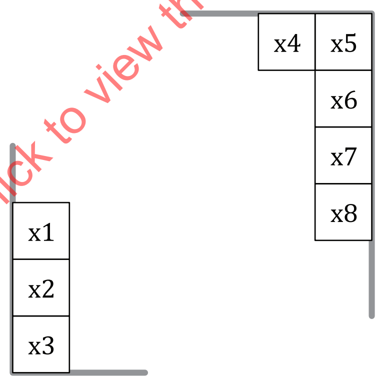


Figure F.6 — Corner condition 4

NOTE 1 Algebraic notation has been used to identify the symbol characters because these vary depending on the symbol format

NOTE 2 The corner characters are identified by the module in the bottom left and top right corners.

F.2.2 Symbol character arrangement

The symbol characters are placed in a matrix in the following manner.

a) A mapping matrix is created.

 1) For small symbols with only one data region, this equates to the mapping matrix.

2) For larger symbols with more than one data region, the mapping matrix equates to an area the size of the abutted data regions. In effect, the mapping matrix has no separating alignment patterns. For example, the 36 x 36 format symbol has four 16 x 16 data regions which abut to create a mapping matrix 32 x 32. The size of the mapping matrix for each symbol format is given in Table 10. The boundary placement case is given in Table F.1.

b) Symbol character 2 is placed in the uppermost left position, with its modules conforming to the bit (or module) sequence defined in Figure 3. Using the notation 2.1 to identify module 1 of symbol character 2, this module is in the top row and leftmost column of every mapping matrix. The module array sequence shown in Figure F.7 is constant for all mapping matrices.

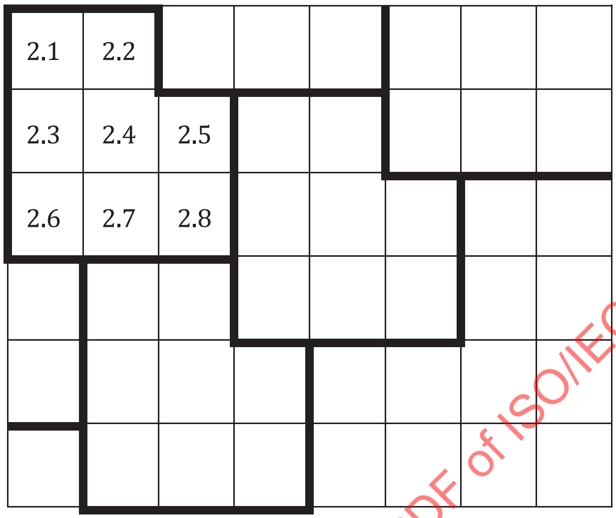


Figure F.7 — Starting sequence for module placement

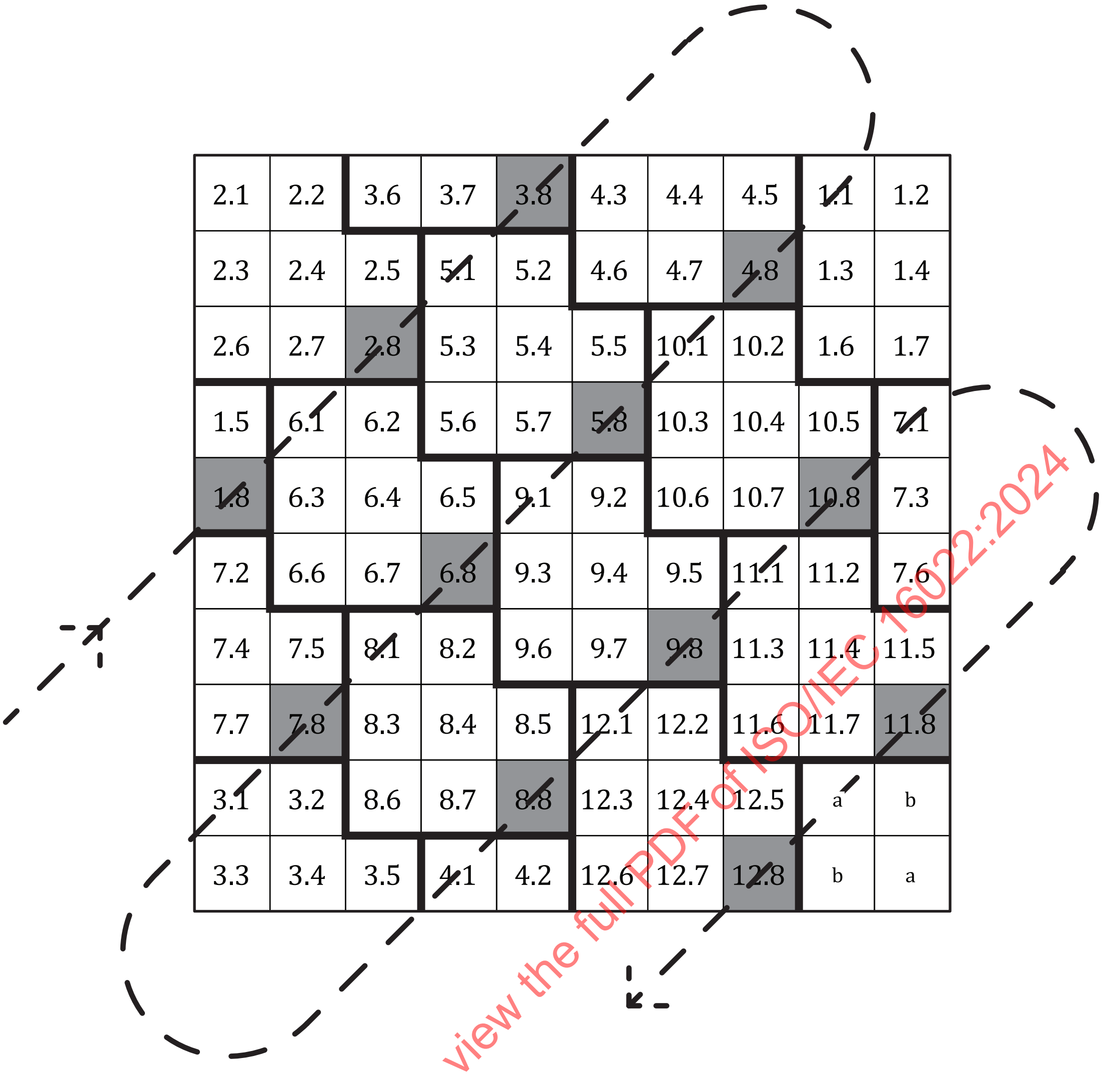
c) The corner shapes are positioned according to Table F.1 and the appropriate Figures F.3 to F.6. Plotting of the standard symbol character shapes continues, nesting the shapes as illustrated in Figure F.7 for symbol characters 2, 5 and 6. The non-standard symbol characters are positioned as per Table F.1. This process results in the mapping matrix being completely covered in symbol characters, most of which are un-numbered.

d) The sequence of symbol characters is determined as follows. Symbol characters are arranged on 45-degree parallel diagonal lines between the lower left and upper right, generally linking through the centres on module 8.

e) The first diagonal line starts with the line through module 8 of symbol character 1; this is module 8 except in the case of the 6 x 28 mapping matrix, where the corner condition, as defined in Figure F.6, determines the values of modules in symbol character 1 (i.e. making the module identified in Figure F.7 as 1.b represent module 1,2). The diagonal line continues through modules 2.8 and 3.8.

f) At this point, the diagonal line crosses the top row boundary. The next diagonal line is started 4 modules to the right in the top row, or in the case of the 8 x 8 mapping matrix, 3 modules right and 1 module down; i.e. the diagonal line is always displaced by 4 modules. Symbol characters are numbered in order, based on the placement path crossing module 8. Thus, the next characters are determined by the downward diagonal line crossing modules 4.8, 5.8, 6.8 and so on.

g) As shown in Figure F.8, the placement path continues as diagonal lines four modules to the right (or four modules down, or combinations thereof) from the previous diagonal line. The first, and all odd numbered, diagonal lines map the symbol character sequence from bottom left to top right. The second, and all even numbered, diagonal lines map the symbol character sequence from the top right to the bottom left.



a BLK.

b WHT.

Figure F.8 — Symbol character placement sequence

h) When the placement path encounters a non-standard symbol character shape, which is not completely contained within the boundaries of the mapping matrix, that symbol character is continued on the opposite side of the matrix. This has the effect of numbering the opposite portions of these symbol characters before the placement path crosses that position. For example, in the illustrated mapping matrix (see Figure F.8) the other portions of symbol character 3 and 7 are pre-numbered before the placement path crosses them. Thus, the placement path only numbers un-numbered symbol characters. These boundary and corner conditions are specified in Table F.1. This can be seen in Figure F.8 for symbol characters 1, 3, 4 and 7. The corner conditions also affect the numbering sequence. The bottom left corner as illustrated in:

— Figure F.3 is numbered immediately before the symbol character above it (see Figures F.11 and F.18 for examples);

— Figure F.4 is numbered immediately before the symbol character above it (see Figures F.12 and F.19 for examples);

— Figure F.5 is numbered immediately after the symbol character to its right (see Figure F.13 for an example);

— Figure F.6 is numbered immediately before the symbol character above it (see Figure F.15 for an example).

The remaining modules of the corner are numbered before the placement path crosses them.

i) The placement procedure continues until all symbol characters are placed, and it ends in the lower right of the mapping matrix. Four sizes of mapping matrix (10 x 10, 14 x 14, 18 x 18 and 22 x 22) have a 2 x 2 area remaining in the bottom right hand corner. The top left and bottom right modules of this area are dark (nominally encoding binary 1). This is illustrated in Figure F.8.

Typical mapping matrices conforming to this procedure are illustrated in F.3. Figures F.9 to F.15 cover respective cases 1 to 7 for boundary placement. Figures F.16 to F.19 are another set of examples for cases 1 to 4. F.1 provides a C language program capable of mapping all encoded bits into the appropriate mapping matrix.

F.3 Symbol character placement examples

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8
2.6	2.7	2.8	5.3	5.4	5.5	1.1	1.2
1.5	6.1	6.2	5.6	5.7	5.8	1.3	1.4
1.8	6.3	6.4	6.5	8.1	8.2	1.6	1.7
7.2	6.6	6.7	6.8	8.3	8.4	8.5	7.1
7.4	7.5	3.1	3.2	8.6	8.7	8.8	7.3
7.7	7.8	3.3	3.4	3.5	4.1	4.2	7.6

Figure F.9 — Codeword placement for square mapping matrix of size 8

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	1.1	1.2
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	1.3	1.4
2.6	2.7	2.8	5.3	5.4	5.5	10.1	10.2	1.6	1.7
1.5	6.1	6.2	5.6	5.7	5.8	10.3	10.4	10.5	7.1
1.8	6.3	6.4	6.5	8.1	8.2	10.6	10.7	10.8	7.3
7.2	6.6	6.7	6.8	8.3	8.4	8.5	11.1	11.2	7.6
7.4	7.5	8.1	8.2	8.6	8.7	8.8	11.3	11.4	11.5
7.7	7.8	8.3	8.4	8.5	12.1	12.2	11.6	11.7	11.8
3.1	3.2	8.6	8.7	8.8	12.3	12.4	12.5	a	b
3.3	3.4	3.5	4.1	4.2	12.6	12.7	12.8	b	a

a BLK.

b WHT.

Figure F.10 — Codeword placement for square mapping matrix of size 10

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	8.4	8.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	8.6
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	8.7
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	14.1	14.2	8.8
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	14.3	14.4	14.5
7.2	6.6	6.7	6.8	11.3	11.4	11.5	15.1	15.2	14.6	14.7	14.8
7.4	7.5	10.1	10.2	11.6	11.7	11.8	15.3	15.4	15.5	1.1	1.2
7.7	7.8	10.3	10.4	10.5	16.1	16.2	15.6	15.7	15.8	1.3	1.4
9.1	9.2	10.6	10.7	10.8	16.3	16.4	16.5	18.1	18.2	1.6	1.7
9.3	9.4	9.5	17.1	17.2	16.6	16.7	16.8	18.3	18.4	18.5	7.1
9.6	9.7	9.8	17.3	17.4	17.5	3.1	3.2	18.6	18.7	18.8	7.3
8.1	8.2	8.3	17.6	17.7	17.8	3.3	3.4	3.5	4.1	4.2	7.6

Figure F.11 — Codeword placement for square mapping matrix of size 12

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	8.4	8.5	8.6	8.7
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	14.1	14.2	8.8
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	14.3	14.4	14.5
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	15.1	15.2	14.6	14.7	14.8
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	15.3	15.4	15.5	1.1	1.2
7.2	6.6	6.7	6.8	11.3	11.4	11.5	16.1	16.2	15.6	15.7	15.8	1.3	1.4
7.4	7.5	10.1	10.2	11.6	11.7	11.8	16.3	16.4	16.5	22.1	22.2	1.6	1.7
7.7	7.8	10.3	10.4	10.5	17.1	17.2	16.6	16.7	16.8	22.3	22.4	22.5	7.1
9.1	9.2	10.6	10.7	10.8	17.3	17.4	17.5	21.1	21.2	22.6	22.7	22.8	7.3
9.3	9.4	9.5	18.1	18.2	17.6	17.7	17.8	21.3	21.4	21.5	23.1	23.2	7.6
9.6	9.7	9.8	18.3	18.4	18.5	20.1	20.2	21.6	21.7	21.8	23.3	23.4	23.5
8.1	19.1	19.2	18.6	18.7	18.8	20.3	20.4	20.5	24.1	24.2	23.6	23.7	23.8
8.2	19.3	19.4	19.5	3.1	3.2	20.6	20.7	20.8	24.3	24.4	24.5	[a]	[b]
8.3	19.6	19.7	19.8	3.3	3.4	3.5	4.1	4.2	24.6	24.7	24.8	[b]	[a]

[a] BLK.

[b] WHT.

Figure F.12 — Codeword placement for square mapping matrix of size 14

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	9.1	9.2	10.6	10.7	10.8	7.3	7.4	7.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	9.3	9.4	9.5	11.1	11.2	7.6	7.7	7.8
2.6	2.7	2.8	5.3	5.4	5.5	8.1	8.2	9.6	9.7	9.8	11.3	11.4	11.5	1.1	1.2
1.5	6.1	6.2	5.6	5.7	5.8	8.3	8.4	8.5	12.1	12.2	11.6	11.7	11.8	1.3	1.4
1.8	6.3	6.4	6.5	3.1	3.2	8.6	8.7	8.8	12.3	12.4	12.5	10.1	10.2	1.6	1.7
7.1	6.6	6.7	6.8	3.3	3.4	3.5	4.1	4.2	12.6	12.7	12.8	10.3	10.4	10.5	7.2

Figure F.13 — Codeword placement for 6 x 16 rectangular mapping matrix

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	11.1	11.2	12.6	12.7	12.8	13.3	13.4	13.5	21.1	21.2	22.6	22.7	22.8	23.3	23.4	23.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	11.3	11.4	11.5	14.1	14.2	13.6	13.7	13.8	21.3	21.4	21.5	24.1	24.2	23.6	23.7	23.8
2.6	2.7	2.8	5.3	5.4	5.5	10.1	10.2	11.6	11.7	11.8	14.3	14.4	14.5	20.1	20.2	21.6	21.7	21.8	24.3	24.4	24.5	1.1	1.2
1.5	6.1	6.2	5.6	5.7	5.8	10.3	10.4	10.5	15.1	15.2	14.6	14.7	14.8	20.3	20.4	20.5	25.1	25.2	24.6	24.7	24.8	1.3	1.4
1.8	6.3	6.4	6.5	9.1	9.2	10.6	10.7	10.8	15.3	15.4	15.5	19.1	19.2	20.6	20.7	20.8	25.3	25.4	25.5	29.1	29.2	1.6	1.7
7.2	6.6	6.7	6.8	9.3	9.4	9.5	16.1	16.2	15.6	15.7	15.8	19.3	19.4	19.5	26.1	26.2	25.6	25.7	25.8	29.3	29.4	29.5	7.1
7.4	7.5	8.1	8.2	9.6	9.7	9.8	16.3	16.4	16.5	18.1	18.2	19.6	19.7	19.8	26.3	26.4	26.5	28.1	28.2	29.6	29.7	29.8	7.3
7.7	7.8	8.3	8.4	8.5	17.1	17.2	16.6	16.7	16.8	18.3	18.4	18.5	27.1	27.2	26.6	26.7	26.8	28.3	28.4	28.5	30.1	30.2	7.6
3.1	3.2	8.6	8.7	8.8	17.3	17.4	17.5	12.1	12.2	18.6	18.7	18.8	27.3	27.4	27.5	22.1	22.2	28.6	28.7	28.8	30.3	30.4	30.5
3.3	3.4	3.5	4.1	4.2	17.6	17.7	17.8	12.3	12.4	12.5	13.1	13.2	27.6	27.7	27.8	22.3	22.4	22.5	23.1	23.2	30.6	30.7	30.8

Figure F.14 — Codeword placement for 10 x 24 rectangular mapping matrix

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	8.1	8.2	9.6	9.7	9.8	10.3	10.4	10.5	14.1	14.2	15.6	15.7	15.8	16.3	16.4	16.5	20.1	20.2	1.4	1.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	8.3	8.4	8.5	11.1	11.2	10.6	10.7	10.8	14.3	14.4	14.5	19.1	19.2	16.6	16.7	16.8	20.3	20.4	20.5	1.6
2.6	2.7	2.8	5.3	5.4	5.5	7.1	7.2	8.6	8.7	8.8	11.3	11.4	11.5	13.1	13.2	14.6	14.7	14.8	19.3	19.4	19.5	19.1	19.2	20.6	20.7	20.8	1.7
1.1	6.1	6.2	5.6	5.7	5.8	7.3	7.4	7.5	12.1	12.2	11.6	11.7	11.8	13.3	13.4	13.5	18.1	18.2	19.6	19.7	19.8	19.3	19.4	19.5	21.1	21.2	1.8
1.2	6.3	6.4	6.5	3.1	3.2	7.6	7.7	7.8	12.3	12.4	12.5	9.1	9.2	13.6	13.7	13.8	18.3	18.4	18.5	25.1	25.2	19.6	19.7	19.8	21.3	21.4	21.5
1.3	6.6	6.7	6.8	3.3	3.4	3.5	4.1	4.2	12.6	12.7	12.8	9.3	9.4	9.5	26.1	26.2	18.6	18.7	18.8	25.3	25.4	25.5	16.1	16.2	21.6	21.7	21.8

Figure F.15 — Codeword placement for 6 x 28 rectangular mapping matrix

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	14.6	14.7	14.8	15.3	15.4	15.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	16.1	16.2	15.6	15.7	15.8
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	16.3	16.4	16.5	1.1	1.2
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	17.1	17.2	16.6	16.7	16.8	1.3	1.4
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	17.3	17.4	17.5	27.1	27.2	1.6	1.7
7.2	6.6	6.7	6.8	11.3	11.4	11.5	18.1	18.2	17.6	17.7	17.8	27.3	27.4	27.5	7.1
7.4	7.5	10.1	10.2	11.6	11.7	11.8	18.3	18.4	18.5	26.1	26.2	27.6	27.7	27.8	7.3
7.7	7.8	10.3	10.4	10.5	19.1	19.2	18.6	18.7	18.8	26.3	26.4	26.5	28.1	28.2	7.6
9.1	9.2	10.6	10.7	10.8	19.3	19.4	19.5	25.1	25.2	26.6	26.7	26.8	28.3	28.4	28.5
9.3	9.4	9.5	20.1	20.2	19.6	19.7	19.8	25.3	25.4	25.5	29.1	29.2	28.6	28.7	28.8
9.6	9.7	9.8	20.3	20.4	20.5	24.1	24.2	25.6	25.7	25.8	29.3	29.4	29.5	8.1	8.2
8.5	21.1	21.2	20.6	20.7	20.8	24.3	24.4	24.5	30.1	30.2	29.6	29.7	29.8	8.3	8.4
8.6	21.3	21.4	21.5	23.1	23.2	24.6	24.7	24.8	30.3	30.4	30.5	32.1	32.2	8.6	8.7
22.2	21.6	21.7	21.8	23.3	23.4	23.5	31.1	31.2	30.6	30.7	30.8	32.3	32.4	32.5	22.1
22.4	22.5	3.1	3.2	23.6	23.7	23.8	31.3	31.4	31.5	14.1	14.2	32.6	32.7	32.8	22.3
22.7	22.8	3.3	3.4	3.5	4.1	4.2	31.6	31.7	31.8	14.3	14.4	14.5	15.1	15.2	22.6

Figure F.16 — Codeword placement for square mapping matrix of size 16

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	14.6	14.7	14.8	15.3	15.4	15.5	1.1	1.2
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	16.1	16.2	15.6	15.7	15.8	1.3	1.4
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	16.3	16.4	16.5	29.1	29.2	1.6	1.7
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	17.1	17.2	16.6	16.7	16.8	29.3	29.4	29.5	7.1
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	17.3	17.4	17.5	28.1	28.2	29.6	29.7	29.8	7.3
7.2	6.6	6.7	6.8	11.3	11.4	11.5	18.1	18.2	17.6	17.7	17.8	28.3	28.4	28.5	30.1	30.2	7.6
7.4	7.5	10.1	10.2	11.6	11.7	11.8	18.3	18.4	18.5	27.1	27.2	28.6	28.7	28.8	30.3	30.4	30.5
7.7	7.8	10.3	10.4	10.5	19.1	19.2	18.6	18.7	18.8	27.3	27.4	27.5	31.1	31.2	30.6	30.7	30.8
9.1	9.2	10.6	10.7	10.8	19.3	19.4	19.5	26.1	26.2	27.6	27.7	27.8	31.3	31.4	31.5	8.1	8.2
9.3	9.4	9.5	20.1	20.2	19.6	19.7	19.8	26.3	26.4	26.5	32.1	32.2	31.6	31.7	31.8	8.3	8.4
9.6	9.7	9.8	20.3	20.4	20.5	25.1	25.2	26.6	26.7	26.8	32.3	32.4	32.5	38.1	38.2	8.6	8.7
8.5	21.1	21.2	20.6	20.7	20.8	25.3	25.4	25.5	33.1	33.2	32.6	32.7	32.8	38.3	38.4	38.5	22.1
8.6	21.3	21.4	21.5	24.1	24.2	25.6	25.7	25.8	33.3	33.4	33.5	37.1	37.2	38.6	38.7	38.8	22.3
22.2	21.6	21.7	21.8	24.3	24.4	24.5	34.1	34.2	33.6	33.7	33.8	37.3	37.4	37.5	39.1	39.2	22.6
22.4	22.5	23.1	23.2	24.6	24.7	24.8	34.3	34.4	34.5	36.1	36.2	37.6	37.7	37.8	39.3	39.4	39.5
22.7	22.8	23.3	23.4	23.5	35.1	35.2	34.6	34.7	34.8	36.3	36.4	36.5	40.1	40.2	39.6	39.7	39.8
3.1	3.2	23.6	23.7	23.8	35.3	35.4	35.5	14.1	14.2	36.6	36.7	36.8	40.3	40.4	40.5	a	b
3.3	3.4	3.5	4.1	4.2	35.6	35.7	35.8	14.3	14.4	14.5	15.1	15.2	40.6	40.7	40.8	b	a

a BLK.

b WHT.

Figure F.17 — Codeword placement for square mapping matrix of size 18

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	14.6	14.7	14.8	15.3	15.4	15.5	32.1	32.2	23.4	23.5
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	16.1	16.2	15.6	15.7	15.8	32.3	32.4	32.5	23.6
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	16.3	16.4	16.5	31.1	31.2	32.6	32.7	32.8	23.7
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	17.1	17.2	16.6	16.7	16.8	31.3	31.4	31.5	33.1	33.2	23.8
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	17.3	17.4	17.5	30.1	30.2	31.6	31.7	31.8	33.3	33.4	33.5
7.2	6.6	6.7	6.8	11.3	11.4	11.5	18.1	18.2	17.6	17.7	17.8	30.3	30.4	30.5	34.1	34.2	33.6	33.7	33.8
7.4	7.5	10.1	10.2	11.6	11.7	11.8	18.3	18.4	18.5	29.1	29.2	30.6	30.7	30.8	34.3	34.4	34.5	1.1	1.2
7.7	7.8	10.3	10.4	10.5	19.1	19.2	18.6	18.7	18.8	29.3	29.4	29.5	35.1	35.2	34.6	34.7	34.8	1.3	1.4
9.1	9.2	10.6	10.7	10.8	19.3	19.4	19.5	28.1	28.2	29.6	29.7	29.8	35.3	35.4	35.5	45.1	45.2	1.6	1.7
9.3	9.4	9.5	20.1	20.2	19.6	19.7	19.8	28.3	28.4	28.5	36.1	36.2	35.6	35.7	35.8	45.3	45.4	45.5	7.1
9.6	9.7	9.8	20.3	20.4	20.5	27.1	27.2	28.6	28.7	28.8	36.3	36.4	36.5	44.1	44.2	45.6	45.7	45.8	7.3
8.5	21.1	21.2	20.6	20.7	20.8	27.3	27.4	27.5	37.1	37.2	36.6	36.7	36.8	44.3	44.4	44.5	46.1	46.2	7.6
8.6	21.3	21.4	21.5	26.1	26.2	27.6	27.7	27.8	37.3	37.4	37.5	43.1	43.2	44.6	44.7	44.8	46.3	46.4	46.5
22.2	21.6	21.7	21.8	26.3	26.4	26.5	38.1	38.2	37.6	37.7	37.8	43.3	43.4	43.5	47.1	47.2	46.6	46.7	46.8
22.4	22.5	25.1	25.2	26.6	26.7	26.8	38.3	38.4	38.5	42.1	42.2	43.6	43.7	43.8	47.3	47.4	47.5	8.1	8.2
22.7	22.8	25.3	25.4	25.5	39.1	39.2	38.6	38.7	38.8	42.3	42.4	42.5	48.1	48.2	47.6	47.7	47.8	8.3	8.4
24.1	24.2	25.6	25.7	25.8	39.3	39.4	39.5	41.1	41.2	42.6	42.7	42.8	48.3	48.4	48.5	50.1	50.2	8.6	8.7
24.3	24.4	24.5	40.1	40.2	39.6	39.7	39.8	41.3	41.4	41.5	49.1	49.2	48.6	48.7	48.8	50.3	50.4	50.5	22.1
24.6	24.7	24.8	40.3	40.4	40.5	3.1	3.2	41.6	41.7	41.8	49.3	49.4	49.5	14.1	14.2	50.6	50.7	50.8	22.3
23.1	23.2	23.3	40.6	40.7	40.8	3.3	3.4	3.5	4.1	4.2	49.6	49.7	49.8	14.3	14.4	14.5	15.1	15.2	22.6

Figure F.18 — Codeword placement for square mapping matrix of size 20

2.1	2.2	3.6	3.7	3.8	4.3	4.4	4.5	13.1	13.2	14.6	14.7	14.8	15.3	15.4	15.5	32.1	32.2	23.4	23.5	23.6	23.7
2.3	2.4	2.5	5.1	5.2	4.6	4.7	4.8	13.3	13.4	13.5	16.1	16.2	15.6	15.7	15.8	32.3	32.4	32.5	33.1	33.2	23.8
2.6	2.7	2.8	5.3	5.4	5.5	12.1	12.2	13.6	13.7	13.8	16.3	16.4	16.5	31.1	31.2	32.6	32.7	32.8	33.3	33.4	33.5
1.5	6.1	6.2	5.6	5.7	5.8	12.3	12.4	12.5	17.1	17.2	16.6	16.7	16.8	31.3	31.4	31.5	34.1	34.2	33.6	33.7	33.8
1.8	6.3	6.4	6.5	11.1	11.2	12.6	12.7	12.8	17.3	17.4	17.5	30.1	30.2	31.6	31.7	31.8	34.3	34.4	34.5	1.1	1.2
7.2	6.6	6.7	6.8	11.3	11.4	11.5	18.1	18.2	17.6	17.7	17.8	30.3	30.4	30.5	35.1	35.2	34.6	34.7	34.8	1.3	1.4
7.4	7.5	10.1	10.2	11.6	11.7	11.8	18.3	18.4	18.5	29.1	29.2	30.6	30.7	30.8	35.3	35.4	35.5	49.1	49.2	1.6	1.7
7.7	7.8	10.3	10.4	10.5	19.1	19.2	18.6	18.7	18.8	29.3	29.4	29.5	36.1	36.2	35.6	35.7	35.8	49.3	49.4	49.5	7.1
9.1	9.2	10.6	10.7	10.8	19.3	19.4	19.5	28.1	28.2	29.6	29.7	29.8	36.3	36.4	36.5	48.1	48.2	49.6	49.7	49.8	7.3
9.3	9.4	9.5	20.1	20.2	19.6	19.7	19.8	28.3	28.4	28.5	37.1	37.2	36.6	36.7	36.8	48.3	48.4	48.5	50.1	50.2	7.6
9.6	9.7	9.8	20.3	20.4	20.5	27.1	27.2	28.6	28.7	28.8	37.3	37.4	37.5	47.1	47.2	48.6	48.7	48.8	50.3	50.4	50.5
8.5	21.1	21.2	20.6	20.7	20.8	27.3	27.4	27.5	38.1	38.2	37.6	37.7	37.8	47.3	47.4	47.5	51.1	51.2	50.6	50.7	50.8
8.6	21.3	21.4	21.5	26.1	26.2	27.6	27.7	27.8	38.3	38.4	38.5	46.1	46.2	47.6	47.7	47.8	51.3	51.4	51.5	8.1	8.2
22.2	21.6	21.7	21.8	26.3	26.4	26.5	39.1	39.2	38.6	38.7	38.8	46.3	46.4	46.5	52.1	52.2	51.6	51.7	51.8	8.3	8.4
22.4	22.5	25.1	25.2	26.6	26.7	26.8	39.3	39.4	39.5	45.1	45.2	46.6	46.7	46.8	52.3	52.4	52.5	58.1	58.2	8.6	8.7
22.7	22.8	25.3	25.4	25.5	40.1	40.2	39.6	39.7	39.8	45.3	45.4	45.5	53.1	53.2	52.6	52.7	52.8	58.3	58.4	58.5	22.1
24.1	24.2	25.6	25.7	25.8	40.3	40.4	40.5	44.1	44.2	45.6	45.7	45.8	53.3	53.4	53.5	57.1	57.2	58.6	58.7	58.8	22.3
24.3	24.4	24.5	41.1	41.2	40.6	40.7	40.8	44.3	44.4	44.5	54.1	54.2	53.6	53.7	53.8	57.3	57.4	57.5	59.1	59.2	22.6
24.6	24.7	24.8	41.3	41.4	41.5	43.1	43.2	44.6	44.7	44.8	54.3	54.4	54.5	56.1	56.2	57.6	57.7	57.8	59.3	59.4	59.5
23.1	42.1	42.2	41.6	41.7	41.8	43.3	43.4	43.5	55.1	55.2	54.6	54.7	54.8	56.3	56.4	56.5	60.1	60.2	59.6	59.7	59.8
23.2	42.3	42.4	42.5	3.1	3.2	43.6	43.7	43.8	55.3	55.4	55.5	14.1	14.2	56.6	56.7	56.8	60.3	60.4	60.5	a	b
23.3	42.6	42.7	42.8	3.3	3.4	3.5	4.1	4.2	55.6	55.7	55.8	14.3	14.4	14.5	15.1	15.2	60.6	60.7	60.8	b	a

a BLK.

b WHT.

Figure F.19 — Codeword placement for square mapping matrix of size 22

Annex G
(normative)

Data Matrix print quality – symbology-specific aspects

G.1 General

Because of differences in symbology structures and reference decode algorithms, the effect of certain parameters on a symbol's reading performance may vary from one symbology to another. ISO/IEC 15415 and ISO/IEC 29158 provide for symbology specifications to define the grading of certain symbology-specific attributes. This annex therefore defines the method of grading Fixed Pattern Damage to be used in the application of ISO/IEC 15415 or ISO/IEC 29158 to Data Matrix.

In case the symbol quality is tested by using ISO/IEC 29158 then all references to ISO/IEC 15415 will be replaced by ISO/IEC 29158. The term “modulation” shall be replaced by “cell modulation”, and the term “MOD” shall be replaced by “CMOD”.

G.2 Interpolation between grade levels

A significant change that is introduced in this edition of this document is the use of “continuous grading”, wherein grades are reported in tenth of a grade level instead of an integer representing the entire grade level as in the previous edition. This reduces troubling fluctuations in grades when small changes in measurements cause a grade to transition between integer grade levels. Grades are to be rounded down to the nearest tenth so that the boundary between whole numbers remains unchanged from the previous edition of this document. Accordingly, it is possible to convert the grade from this edition to the grade from the previous edition by simply dropping the decimal part of the grade in all cases.

G.3 Data Matrix Fixed Pattern Damage

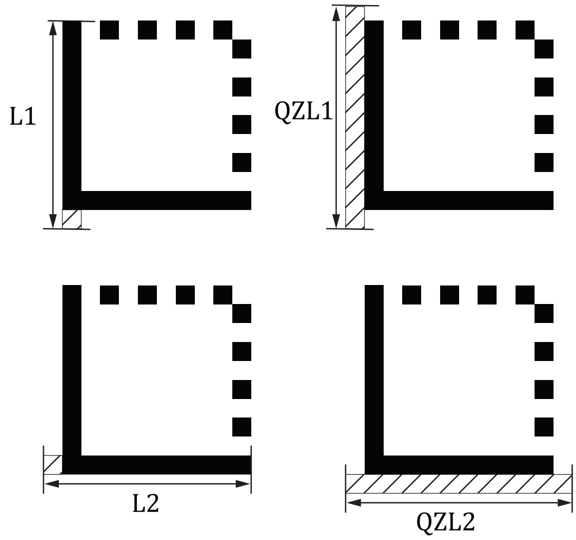
G.3.1 Features to be assessed

The fixed pattern features to be assessed are contained in the one-module wide perimeter of the symbol and the quiet zone of a minimum of one module width (or more if specified by the application) surrounding the symbol. In larger symbols (square symbols 32 x 32 modules or larger, or rectangular symbols 8 x 32 or 12 x 36 or larger) with internal alignment patterns, the alignment pattern is also part of the fixed pattern. The left and lower side of the symbol should form a one-module wide solid "L" shape and the right and upper sides should consist of alternating dark and light single modules (known as the clock track). The alignment bars and internal clock track of the alignment pattern should similarly be a one-module wide solid bar or a series of alternating dark and light single modules respectively. The grading of Fixed Pattern Damage takes account not only of the total number of damaged modules but also of concentrations of damage.

G.3.2 Grading of the outside L of the fixed pattern

Damage to each side of the L shall be graded based on the modulation of the individual modules that compose it. These measurements are applied to the full length of the L sides and to the associated quiet zones.

Figure G.1 below indicates the four segments L1, L2, QZL1 and QZL2. Segment L1 is the vertical portion of the L and extends to the module in the quiet zone adjacent to the L corner. Segment L2 is the horizontal portion of the L and extends to the module in the quiet zone adjacent to the L corner. Segments QZL1 and QZL2 are the portions of the quiet zone adjacent to L1 and L2 respectively and extend one module beyond the end of L1 and L2 respectively, furthest from the corner and are shown shaded in Figure G.1. The corner module at the intersection of L1 and L2 is included in both segments, as is that at the intersection of QZL1 and QZL2.



L1 vertical portion of the L

L2 horizontal portion of the L

QZL1 portion of the quiet zone adjacent to L1

QZL2 portion of the quiet zone adjacent to L2

Figure G.1 — Outside L and corresponding quiet zone segments of fixed pattern

The procedure described below shall be applied to each segment in turn.

a) Find the modulation grade for each module based on the values in ISO/IEC 15415. Since the intended light or dark nature of the module is known, any module sampled to be the opposite colour shall be given modulation grade 0.

b) For each modulation grade level apply the parameter grade overlay technique described in ISO/IEC 15415:

   1) For each side of the L (L1 and L2 in Figure G.1) and each quiet zone area (QZL1 and QZL2, adjacent to L1 and L2 respectively in Figure G.1), assume that all modules not achieving that grade or a higher grade are module errors, and derive a notional damage grade based on the grade thresholds shown in Table G.1. Take the lower of the modulation grade level and the notional damage grade.

   2) The grade for each segment shall be the highest resulting grade for all modulation grade levels.

c) Additionally, for both square and rectangular symbols with more than one data region, repeat steps a) and b) above where L1 and L2 start with the module in the quiet zone and end at the module in the clock track area of the same data region, and QZL1 and QZL2 consist of the quiet zone adjacent to these L1 and L2 segments as defined like Figure G.1. In other words, treat the lower left data region as if it were a symbol with a single data region. If this grade is lower than that obtained from L1, L2, QZL1, and QZL2 in steps a) and b) then replace the grade obtained in steps a) and b) with this grade.

d) Additionally, for segments L1 and L2, verify that all gaps are separated by at least four correct modules and that no gaps are wider than three modules; if this test fails, the grade obtained from the above steps shall be reduced to 0 at that modulation grade level. Take the lower of the modulation grade level and the notional damage grade.

Table G.1 — Grade thresholds for notional damage

Percentage of modules damaged	Grade
0 %	4
> 0 % and ≤ 9 %	3,0 to 3,9
> 9 % and ≤ 13 %	2,0 to 2,9
> 13 % and ≤ 17 %	1,0 to 1,9
> 17 % and ≤ 21 %	0,0 to 0,9
> 21 %	0

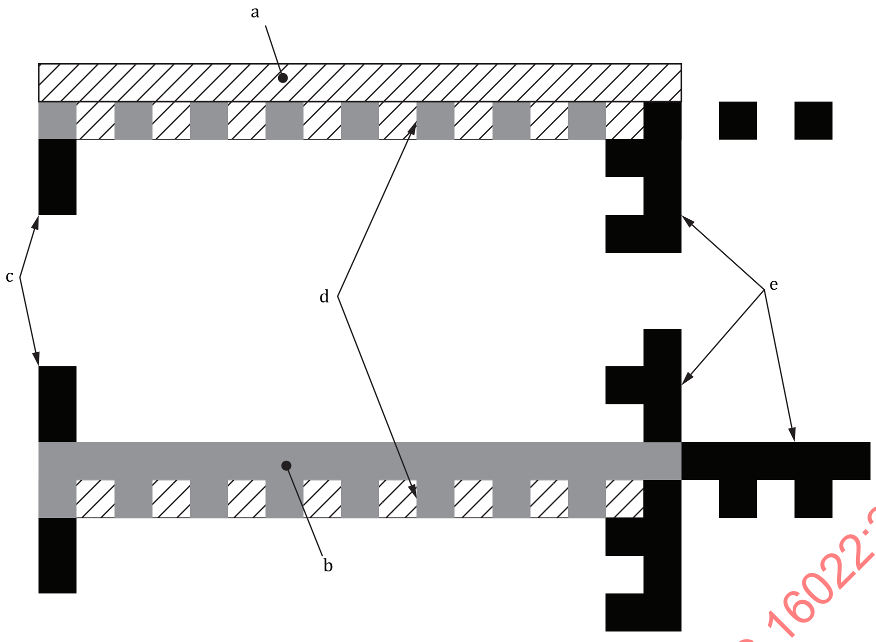
e) The grade for the segment shall be the highest resulting grade for all modulation grade levels.

G.3.3 Grading of the clock track and adjacent solid area segments

This section defines the measurement of damage to the internal alignment patterns (when present) and also external clock tracks and associated quiet zone areas. These tests are applied separately to each segment of the internal alignment patterns, the clock tracks, and associated quiet zone areas that bound the data region, or individual data regions of larger symbols. Each segment consists of a clock track portion and a solid area portion (which is part either of the quiet zone or of an internal alignment bar).

A clock track segment commences with a module in the L side or internal alignment pattern. It continues to and includes the module preceding either the quiet zone or the next internal alignment pattern. The adjacent module in the quiet zone, or the alignment pattern, is not included in the clock track (see Figure G.2).

A solid area segment starts adjacent to the first module of the associated clock track segment. If the other end of the segment is adjacent to a quiet zone, then this solid area segment ends with the module adjacent to the last module of the associated clock track segment (see Figure G.6). If not, it ends with the first module of the adjacent solid segment, which has the same colour (see Figure G.5).



a Solid area (light).
b Solid area (dark).
c L side.
d Clock track segment.
e Alignment bar.

NOTE Figure G.2 depicts an internal alignment pattern segment which terminates at another internal alignment segment of the same colour.

Figure G.2 — Structure of external clock track segment and internal alignment pattern segment

For each external clock track segment or internal alignment pattern segment of a symbol (for multi-segment symbols), damage is measured according to the following procedure.

a) Transition ratio test: on every clock track segment in the binarised image, both external (adjacent to the quiet zone) and internal (adjacent to the solid internal alignment bar), count the number of transitions in the clock track side, Tc, and the solid line side, Ts, and compute and grade the transition ratio using Formula (G.1) and Formula (G.2):

$$Ts' = \text{Max}\,(0, Ts - 1) \quad \text{(G.1)}$$

$$R_T = Ts' / Tc \quad \text{(G.2)}$$

where R_T is the transition ratio.

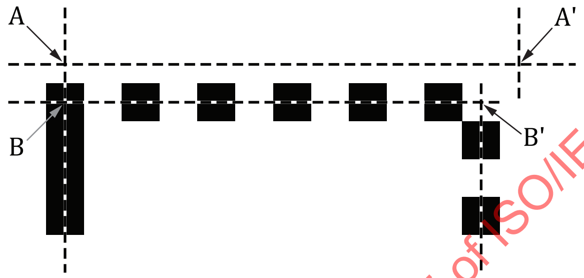
Grade transition ratio according to Table G.2.

Table G.2 — Grading of transition ratio

R_T (transition ratio)	Grade
≤ 0,06	4
> 0,06 and ≤ 0,095	3,0 to 3,9
> 0,095 and ≤ 0,13	2,0 to 2,9
> 0,13 and ≤ 0,165	1,0 to 1,9
> 0,165 and ≤ 0,2	0,0 to 0,9
> 0,2	0

The end points between which transitions are counted are the intersections of grid lines plotted by the reference decode algorithm in the first and last modules of the clock track or solid area.

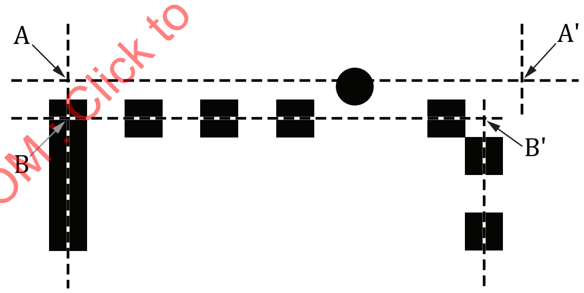
As an example, a perfect symbol with 0 transitions in the quiet zone is depicted in Figure G.3, while a damaged symbol with 2 transitions in the quiet zone is depicted in Figure G.4.



Key

--- grid linking module centres from reference decode algorithm
A - A' intersections defining beginning and end of solid area
B - B' intersections defining beginning and end of clock track

Figure G.3 — Transitions in a perfect symbol



Key

--- grid linking module centres from reference decode algorithm
A - A' intersections defining beginning and end of solid area
B - B' intersections defining beginning and end of clock track

Figure G.4 — Transitions in a damaged symbol

b) Notional damage grade: find the modulation grade for each module based on the values in ISO/IEC 15415. Since the intended light or dark nature of the module is known, any module intended to be dark but the reflectance of which is above the global threshold, and any module intended to be light but the reflectance of which is below the global threshold shall be given modulation grade 0.

c) For each modulation grade level: assume that all modules not achieving that grade or a higher grade are module errors, and derive a notional damage grade based on the following three assessments:

d) Clock track regularity test: for each segment of clock track, taking groups of five adjacent modules and progressing along the segment in steps of one module, verify that in any group of five adjacent modules no more than two are module errors; if this condition is met, the clock track regularity grade shall be 4, otherwise it shall be 0.

e) Clock track damage test: for each segment, count the number of incorrect modules in the clock track for the segment; the percentage *P* of incorrect modules over the length of the area shall result in the percentage damage grades shown in Table G.3.

f) Solid fixed pattern test: for each segment, count the number of incorrect modules in the solid area (internal alignment bar or external quiet zone area) adjacent to the clock track; the percentage *P* of incorrect modules over the length of the area shall result in the percentage damage grades shown in Table G.3.

Table G.3 — Grading of percentage damage to clock track segments and solid area segments

P	Grade
≤ 10 %	4
> 10 % and ≤ 15 %	3,0 to 3,9
> 15 % and ≤ 20 %	2,0 to 2,9
> 20 % and ≤ 25 %	1,0 to 1,9
> 25 % and ≤ 30 %	0,0 to 0,9
> 30 %	0

g) At each grade level take the lowest of the modulation grade level, the clock track regularity grade, the clock track percentage damage grade, and the solid fixed pattern percentage damage grade.

h) The notional damage grade for the segment shall be the highest resulting grade for all modulation grade levels.

i) The fixed pattern damage grade for the segment shall be the lower of the transition ratio grade and the notional damage grade.

j) The overall fixed pattern damage grade for the clock track and adjacent solid area segments is the lowest of the grades obtained for each of the individual segments.

The shaded areas in Figure G.5 and G.6 show examples of internal alignment pattern segments, which include the clock track portion and solid area portion to which the transition ratio, regularity and solid fixed pattern tests are applied for each type of internal segment boundaries.