

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

ICS 01.080.50; 35.040

National foreword

This British Standard is the UK implementation of ISO/IEC 16022:2006, incorporating corrigenda October 2008 and February 2011. It supersedes BS ISO/IEC 16022:2000 which is withdrawn.

The start and finish of text introduced or altered by corrigendum is indicated in the text by tags. Text altered by ISO/IEC corrigendum October 2008 is indicated in the text by AC1 AC1. Text altered by ISO/IEC corrigendum February 2011 is indicated in the text by AC2 AC2.

The UK participation in its preparation was entrusted by Technical Committee IST/34, Automatic identification and data capture techniques.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 October 2006

© BSI 2011

ISBN 978 0 580 73412 0

Amendments/corrigenda issued since publication

Date	Comments
30 June 2009	Implementation of ISO/IEC corrigendum October 2008
30 June 2011	Implementation of ISO/IEC corrigendum February 2011

INTERNATIONAL STANDARD

ISO/IEC 16022

Second edition
2006-09-15

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

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

Reference number
ISO/IEC 16022:2006(E)



Documents
Document Sharing Hub

Contents

Page

Foreword.....	vii
Introduction	viii
1 Scope	1
2 Normative references	1
3 Terms, definitions, symbols and abbreviated terms, and mathematical/logical notations	2
3.1 Terms and definitions.....	2
3.2 Symbols and abbreviations	2
3.3 Mathematical/logical notations	3
4 Symbol description.....	3
4.1 Basic characteristics	3
4.2 Summary of additional features	4
4.3 Symbol structure	4
4.3.1 Finder pattern	5
4.3.2 Symbol sizes and capacities	5
5 ECC 200 requirements.....	5
5.1 Encode procedure overview	5
5.2 Data encodation	6
5.2.1 Overview	6
5.2.2 Default character interpretation	6
5.2.3 ASCII encodation	7
5.2.4 Symbology control characters	7
5.2.5 C40 encodation	9
5.2.6 Text encodation	11
5.2.7 ANSI X12 encodation	11
5.2.8 EDIFACT encodation	12
5.2.9 Base 256 encodation	12
5.3 User considerations	13
5.3.1 User selection of Extended Channel Interpretation	13
5.3.2 User selection of symbol size and shape	13
5.4 Extended Channel Interpretation	13
5.4.1 Encoding ECIs.....	14
5.4.2 ECIs and Structured Append	15
5.4.3 Post-decode protocol	15
5.5 ECC 200 symbol attributes	15
5.5.1 Symbol sizes and capacity	15
5.5.2 Insertion of Alignment Patterns into larger symbols	17
5.6 Structured Append	17
5.6.1 Basic principles	17
5.6.2 Symbol sequence indicator	17
5.6.3 File identification	18
5.6.4 FNC1 and Structured Append	18
5.6.5 Buffered and unbuffered operation	18
5.7 Error detection and correction	18
5.7.1 Reed-Solomon error correction	18
5.7.2 Generating the error correction codewords	18
5.7.3 Error correction capacity	19
5.8 Symbol construction	20
5.8.1 Symbol character placement.....	20
5.8.2 Alignment Pattern module placement	20

5.8.3	Finder Pattern module placement	20
6	ECC 000 - 140 requirements.....	21
6.1	Use recommendations.....	21
6.2	Encode procedure overview	21
6.3	Data encodation	21
6.3.1	Base 11 - Numeric encodation.....	23
6.3.2	Base 27 - Upper-case Alphabetic encodation	23
6.3.3	Base 37 - Upper-case Alphanumeric encodation	23
6.3.4	Base 41 - Upper-case Alphanumeric plus Punctuation encodation	24
6.3.5	ASCII encodation	24
6.3.6	8-bit byte encodation	24
6.4	User selection of error correction level	24
6.4.1	Selection of error correction level.....	24
6.4.2	Other error correction levels based on convolutional code algorithms	25
6.5	Constructing the Unprotected Bit Stream	25
6.5.1	Format ID Bit Field	25
6.5.2	CRC Bit Field	25
6.5.3	Data Length Bit Field	25
6.5.4	Data prefix construction.....	25
6.5.5	Completing the Unprotected Bit Stream.....	26
6.6	Constructing the Unrandomised Bit Stream	26
6.6.1	Header construction	26
6.6.2	Applying convolutional coding to create the Protected Bit Stream	26
6.6.3	Trailer construction	27
6.6.4	Completing the Unrandomised Bit Stream.....	27
6.7	Pattern randomising	27
6.8	Module placement in matrix.....	27
7	Symbol dimensions	27
7.1	Dimensions	27
7.2	Quiet zone	27
8	Symbol quality.....	27
8.1	Symbol quality parameters	28
8.1.1	Fixed pattern damage	28
8.1.2	Scan grade and overall symbol grade	28
8.1.3	Grid non-uniformity	28
8.2	Process control measurements.....	28
9	Reference decode algorithm for Data Matrix	28
10	User guidelines	38
10.1	Human readable interpretation.....	38
10.2	Autodiscrimination capability.....	38
10.3	System considerations.....	38
11	Transmitted data	38
11.1	Protocol for FNC1 (ECC 200 only)	38
11.2	Protocol for FNC1 in the second position (ECC 200 only)	38
11.3	Protocol for Macro characters in the first position (ECC 200 only).....	38
11.4	Protocol for ECIs (ECC 200 only)	39
11.5	Symbology identifier.....	39
11.6	Transmitted data example.....	39
Annex A	(normative) ECC 200 interleaving process.....	40
A.1	Schematic illustration	40
A.2	Starting sequence for interleaving in different sized symbols	40
Annex B	(normative) ECC 200 pattern randomising.....	43
B.1	253-state algorithm	43
B.1.1	253-state randomising algorithm.....	43
B.1.2	253-state un-randomising algorithm	43
B.2	255-state algorithm	44

B.2.1	255-state randomising algorithm	44
B.2.2	255-state un-randomising algorithm.....	44
Annex C	(normative) ECC 200 encodation character sets.....	45
C.1	C40 encodation character set.....	45
C.2	Text encodation character set.....	46
C.3	EDIFACT encodation character set.....	47
Annex D	(normative) ECC 200 alignment patterns	48
Annex E	(normative) ECC 200 Reed-Solomon error detection and correction	50
E.1	Error correction codeword generator polynomials.....	50
E.2	Error correction calculation.....	52
E.3	Calculation of error correction codewords.....	53
Annex F	(normative) ECC 200 symbol character placement	55
F.1	Symbol character placement.....	55
F.2	Symbol character placement rules	57
F.2.1	Non-standard symbol character shapes	57
F.2.2	Symbol character arrangement.....	60
F.3	Symbol character placement examples for ECC 200.....	63
Annex G	(normative) ECC 000 - 140 symbol attributes	68
G.1	ECC 000	68
G.2	ECC 050	69
G.3	ECC 080	70
G.4	ECC 100	71
G.5	ECC 140	72
Annex H	(normative) ECC 000 - 140 data module placement grids	73
Annex I	(normative) ECC 000 - 140 character encodation schemes.....	90
I.1	Base 11 encodation scheme.....	94
I.1.1	First stage procedure	94
I.1.2	Second stage procedure.....	94
I.1.3	Example	94
I.2	Base 27 encodation scheme.....	95
I.2.1	First stage procedure	95
I.2.2	Second stage procedure.....	95
I.2.3	Example	95
I.3	Base 37 encodation scheme.....	96
I.3.1	First stage procedure	96
I.3.2	Second stage procedure.....	96
I.3.3	Example	96
I.4	Base 41 encodation scheme.....	97
I.4.1	First stage procedure	97
I.4.2	Second stage procedure.....	97
I.4.3	Example	97
Annex J	(normative) ECC 000 - 140 CRC algorithm	98
J.1	CRC state machine	98
J.2	CRC polynomial	98
J.3	CRC 2-byte header.....	98
Annex K	(normative) ECC 000 - 140 error checking and correcting algorithms	100
K.1	ECC 000	100
K.2	ECC 050	100
K.3	ECC 080	100
K.4	ECC 100	100
K.5	ECC 140	100
K.6	Processing the convolutional code	100
K.7	Convolutional codes reference decode algorithm	101
Annex L	(normative) ECC 000 - 140 Master Random Bit Stream (in hexadecimal).....	104

Annex M (normative) Data Matrix print quality – symbology-specific aspects	105
M.1 Data Matrix Fixed Pattern Damage	105
M.1.1 Features to be assessed	105
M.1.2 Grading of the outside L of the fixed pattern	105
M.1.3 Grading of the clock track and adjacent solid area segments	107
M.1.4 Calculation and grading of average grade	111
M.2 Scan grade	112
Annex N (normative) Symbology identifier	113
Annex O (informative) ECC 200 encode example	114
Annex P (informative) Encoding data using the minimum symbol data characters for ECC 200	116
Annex Q (informative) ECC 000 - 140 encode example using ECC 050	120
Q.1 Encode example	120
Q.2 CRC calculation for example	125
Annex R (informative) Useful process control techniques	128
R.1 Symbol contrast	128
R.2 Special reference symbol	128
R.3 Assessing Axial Nonuniformity	129
R.4 Visual inspection for symbol distortion and defects	129
Annex S (informative) Autodiscrimination capability	130
Annex T (informative) System considerations	131
Bibliography	132

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 16022 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

This second edition cancels and replaces the first edition (ISO/IEC 16022:2000), which has been technically revised. It also incorporates the Technical Corrigendum ISO/IEC 16022:2000/Cor.1:2004.

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 International Standard 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 require 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 International Standard 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 referenced documents are indispensable for the application 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 15424, *Information technology — Automatic identification and data capture techniques — Data Carrier Identifiers (including Symbology Identifiers)*

ISO/IEC 19762-1, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 1: General terms relating to AIDC*

ISO/IEC 19762-2, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 2: Optically readable media (ORM)*

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

ISO/IEC 15416, *Information technology — Automatic identification and data capture techniques — Bar code print quality test specification — Linear symbols*

ISO/IEC 646:1991, *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 8859-5:1999, *Information technology — 8-bit single-byte coded graphic character sets — Part 5: Latin/Cyrillic alphabet*

AIM Inc. ITS/04-001 International Technical Standard: *Extended Channel Interpretations — Part 1: Identification Schemes and Protocol*

3 Terms, definitions, symbols and mathematical/logical notations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762-1, ISO/IEC 19762-2 and the following apply.

3.1.1

codeword

symbol character value, an intermediate level of coding between source data and the graphical encodation in the symbol

3.1.2

module

single cell in a matrix symbology used to encode one bit of data, nominally a square shape in Data Matrix

3.1.3

convolutional coding

error checking and correcting (ECC) algorithm that processes a set of input bits into a set of output bits that can recover from damage by breaking the input bits into blocks, then convolving each input block with the contents of a multi-stage shift register to produce protected output blocks

NOTE These encoders can be constructed in hardware using input and output switches, shift registers, and exclusive-or (XOR) gates.

3.1.4

pattern randomising

procedure to convert an original bit pattern to another bit pattern, intended to reduce the probability of repeating patterns occurring in the symbol, by inverting selected bits

3.2 Symbols

For the purposes of this document, the following mathematical symbols apply unless defined locally.

d number of error correction codewords

e number of erasures

k (for ECC 000 - 140) the number of bits in a complete segment input to the state machine to generate the convolutional code (for ECC 200) total number of error correction codewords

m the memory order of the convolutional code

n (for ECC 000 - 140) the number of bits in a complete segment generated by the state machine producing the convolutional code (for ECC 200) total number of data codewords

N the numerical base in an encodation scheme

p number of codewords reserved for error detection

S symbol character

t number of errors

u the input bit segment to the state machine, taken k bits at a time

v the output bit segment from the state machine, generated n bits at a time

X horizontal and vertical width of a module

ε error correction codeword

3.3 Mathematical/logical notations

For the purposes of this document, the following notations and mathematical operations apply.

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.

LSB least significant bit

MSB most significant bit

4 Symbol description

4.1 Basic characteristics

Data Matrix is a two-dimensional matrix symbology.

There are two types:

ECC 200 which uses Reed-Solomon error correction. ECC 200 is recommended for new applications.

ECC 000 - 140 with several available levels of convolutional error correction, referred to as ECC 000, ECC 050, ECC 080, ECC 100 and ECC 140 respectively. ECC 000 - 140 should only be used in closed applications where a single party controls both the production and reading of the symbols and is responsible for overall system performance.

The characteristics of Data Matrix are:

a) Encodable character set:

- 1) values 0 – 127 in accordance with the US national version of ISO/IEC 646

NOTE 1 This version consists of the G0 set of ISO/IEC 646 and the C0 set of ISO/IEC 6429 with values 28 – 31 modified to FS, GS, RS and US respectively.

- 2) values 128 - 255 in accordance with ISO 8859-1. These are referred to as extended ASCII.

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

NOTE 2 This International Standard specifies Data Matrix symbols in terms of dark modules marked on a light background. However, subclause 4.2 provides that symbols may also be produced with light and dark modules reversed in colour (see 4.2), and in such symbols references in this International Standard to dark modules should be taken as references to light modules, and vice versa.

c) Symbol size in modules (not including quiet zone):

ECC 200 10 x 10 to 144 x 144 even values only

ECC 000 – 140 9 x 9 to 49 x 49, odd values only

- d) Data characters per symbol (for maximum symbol size in ECC200):
 - 1) Alphanumeric data: up to 2 335 characters
 - 2) 8-bit byte data: 1 555 characters
 - 3) Numeric data: 3 116 digits.
- e) Selectable error correction:
 - ECC 200: Reed-Solomon error correction.
 - ECC 000 - 140: Four levels of convolutional error correction, plus the option to apply only error detection
- f) Code type: Matrix
- g) Orientation independence: Yes

4.2 Summary of additional features

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

- a) Reflectance reversal: (Inherent): Symbols are intended to be read when marked so that the image is either dark on light or light on dark (see Figure 1). The specifications in this International Standard 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: (ECC 200 only, optional): 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: (ECC 200 only, optional): Six symbol formats are specified in a rectangular form.
- d) Structured append: (ECC 200 only, 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.

4.3 Symbol structure

Each Data Matrix symbol consists of data regions which contain nominally square modules set out in a regular array. In larger ECC 200 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 an ECC 140 and two representations of an ECC 200 symbol.

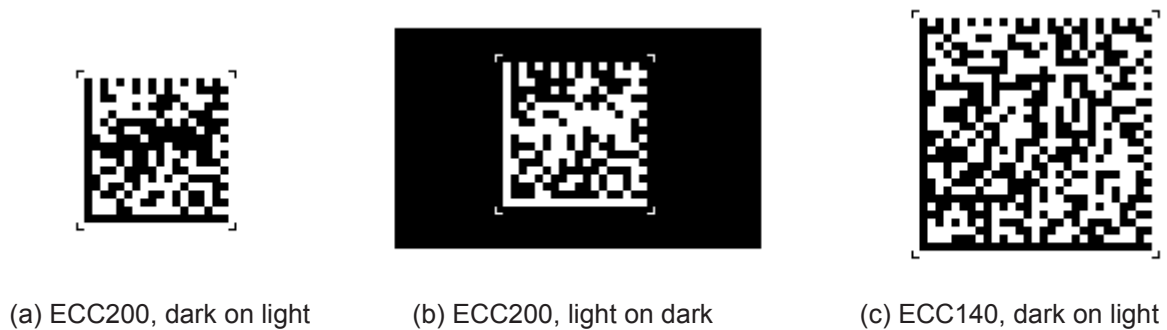


Figure 1 — ECC 200 (a & b) and ECC 140 (c) encoding "A1B2C3D4E5F6G7H8I9J0K1L2"

4.3.1 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 dark 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.

4.3.2 Symbol sizes and capacities

ECC 200 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. All ECC 200 symbols can be recognised by the upper right corner module being light. The complete attributes of ECC 200 symbols are given in Table 7 in Section 5.5.

ECC 000 - 140 symbols have an odd number of rows and an odd number of columns. Symbols are square with sizes from 9 x 9 to 49 x 49 (modules) not including quiet zones. These symbols can be recognised by the upper right corner module being dark. The complete attributes of ECC 000 - 140 symbols are given in Annex G.

5 ECC 200 requirements

5.1 Encode procedure overview

This section provides an overview of the encoding procedure. Following sections will provide more details. An encoding example for ECC 200 is given in Annex O. The following steps convert user data to an ECC 200 symbol:

Step 1: Data encodation

Analyse the data stream to identify the variety of different characters to be encoded. ECC 200 includes various encodation schemes which allow a defined set of characters to be converted into codewords more efficiently than the default scheme. 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 Section 5.5, Table 7.

Table 1 — Encodation schemes for ECC 200

Encodation scheme	Characters	Bits per data character
ASCII	double digit numerics	4
	ASCII values 0 - 127	8
	Extended ASCII values 128 - 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 - 94	6
Base 256	All byte values 0 - 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		

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.

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.

5.2 Data encodation

5.2.1 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 P). It should also be noted that even if the number of codewords is minimised, the codeword stream might need to be expanded to fill a symbol. This fill process is done using pad characters.

5.2.2 Default character interpretation

The default character interpretation for character values 0 to 127 shall conform to ISO/IEC 646. The default character interpretation for character values 128 to 255 shall conform to ISO 8859-1: Latin Alphabet No. 1. The graphical representation of data characters shown throughout this document complies with the default interpretation. This interpretation can be changed using Extended Channel Interpretation (ECI) escape sequences, see 5.4. The default interpretation corresponds to ECI 000003.

5.2.3 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 5.2.4.2). 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.

Table 2 — ASCII encodation values

Codeword	Data or function
1 - 128	ASCII data (ASCII value + 1)
129	Pad
130 - 229	2-digit data 00 - 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
237	06 Macro
238	Latch to ANSI X12 encodation
239	Latch to Text encodation
240	Latch to EDIFACT encodation
241	ECI Character
242 - 255	Not to be used in ASCII encodation

5.2.4 Symbology control characters

ECC 200 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 5.2.4.1 to 5.2.4.9. These symbology control characters, with the exception of values from 242 through 255, are found in the ASCII encodation set (see Table 2).

5.2.4.1 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 set.

5.2.4.2 Upper Shift character

The Upper Shift character is used in combination with an ASCII value (1 - 128) to encode an extended ASCII character (129-255). 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 - 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.

5.2.4.3 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 is applied to the pad characters starting at the second pad character and continuing to the end of the symbol (see Annex B.1).

5.2.4.4 Extended Channel Interpretation character

An Extended Channel Interpretation (ECI) character is used to change from the default interpretation used to encode data. The Extended Channel Interpretation protocol is common across a number of symbologies and its application to ECC 200 is defined more fully in 5.4. 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.

5.2.4.5 Shift characters in C40 and Text encodation

In C40 and Text Encoding, 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.

5.2.4.6 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).

5.2.4.7 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 must 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	$[>^R_s 05^G_s]$	$^R_s E_{OT}$
237	06 Macro	$[>^R_s 06^G_s]$	$^R_s E_{OT}$

5.2.4.8 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 5.6.

5.2.4.9 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.

5.2.5 C40 encodation

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-Z and 0-9. They are assigned to a single C40 values. Characters of the other sets are assigned to one of the three shift characters, pointing to one of the 3 remaining subset, followed by one of the C40 values (see Annex C, 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 5.2.5.2.). 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.

5.2.5.1 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.

5.2.5.2 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:

$$(1600 * C1) + (40 * C2) + C3 + 1$$

which produces a value from 1 to 64000. Figure 2 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 must represent a data character); encode the two remaining C40 values followed by a pad C40 value of 0 (Shift 1) in the last two symbol characters. 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.

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	$23307 \text{ div } 256 = 91$
2nd codeword: (16-bit value) mod 256	$23307 \text{ mod } 256 = 11$
Codewords	91, 11

Figure 2 — Example of C40 encoding

5.2.5.3 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 Annex C.1; V has been used to indicate the appropriate C40 value.

5.2.6 Text encodation

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 are shown in Annex C, Table C.2.

5.2.6.1 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.

5.2.6.2 Text encodation rules

The rules for C40 encodation apply.

5.2.7 ANSI X12 encodation

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 4. There are no shift characters in the ANSI X12 encodation set.

Table 4 — 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 - 13	0 - 9	48 - 57
14 - 39	A - Z	65 - 90

5.2.7.1 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.

5.2.7.2 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.

5.2.8 EDIFACT encodation

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.

5.2.8.1 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.

5.2.8.2 EDIFACT encodation Rules

The EDIFACT encodation character set is defined in Annex C, 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 Figure 3. 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 Figure 4. When EDIFACT encodation is terminated with the Unlatch character, any remaining bits left in the single symbol character shall be filled with zeros. ASCII mode 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.

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.			

Figure 3 — The relationship between the EDIFACT value and the 8-bit byte value

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					

Figure 4 — Example of EDIFACT encodation

5.2.9 Base 256 encodation

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 5.2.2. The 255-State pattern randomising algorithm is 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.

5.2.9.1 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.

5.2.9.2 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 5 specifies how the field length is defined. Thereafter, all encodation shall be of the byte values.

Table 5 — Base 256 field length

Field Length	Values of $d1$, $d2$	Permitted Values of d
Remainder of Symbol	$d1 = 0$	$d1 = 0$
1 to 249	$d1 = \text{length}$	$d1 = 1 \text{ to } 249$
250 to 1555	$d1 = (\text{length DIV } 250) + 249$	$d1 = 250 \text{ to } 255$
	$d2 = \text{length MOD } 250$	$d2 = 0 \text{ to } 249$

5.3 User considerations

ECC 200 offers flexibility in the way data is encoded. Alternate character sets may be invoked using the ECI protocol. Data may be encoded in square or rectangular symbols. Where the message length exceeds the capacity of a single symbol, it is also possible to encode it in a Structured Append sequence of up to 16 separate but logically linked ECC 200 symbols (see 5.6).

5.3.1 User selection of Extended Channel Interpretation

The use of an alternative Extended Channel Interpretation to identify a particular code page or more specific data interpretation requires additional codewords to invoke the feature. The use of the Extended Channel Interpretation protocol (see 5.4) provides the capability to encode data from alphabets other than the Latin alphabet (ISO 8859-1 Latin Alphabet No. 1) supported by the default interpretation (ECI 000003).

5.3.2 User selection of symbol size and shape

ECC 200 has twenty-four square and six rectangular symbol configurations. The size and shape may be selected to suit the requirement of the application. These configurations are technically specified in 5.5.

5.4 Extended Channel Interpretation

The Extended Channel Interpretation (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 Extended Channel Interpretation protocol is fully specified in AIM Inc. International Technical Specification – Extended Channel Interpretations Part 1. The protocol provides a consistent method to specify particular interpretations on byte values before printing and after decoding. The Extended Channel Interpretation 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 Extended Channel Interpretation 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.

The Extended Channel Interpretation protocol shall only be applied to ECC 200 symbols. A specified Extended Channel Interpretation may be invoked anywhere in the encoded message.

5.4.1 Encoding ECIs

The various encodation schemes of Data Matrix for ECC 200 (defined in Table 1) may be applied under any of the Extended Channel Interpretations. 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 6.

The following examples illustrate the encodation:

ECI = 015000

Codewords:

$$\begin{aligned} & [241] [(15000 - 127) \text{ div } 254 + 128] [(15000 - 127) \text{ mod } 254 + 1] \\ &= [241] [58 + 128] [141 + 1] \\ &= [241] [186] [142] \end{aligned}$$

ECI = 090000

Codewords:

$$\begin{aligned} & [241] [(90000 - 16383) \text{ div } 64516 + 192] [((90000 - 16383) \text{ div } 254) \text{ mod } 254 + 1] [(90000 - 16383) \text{ mod } 254 + 1] \\ &= [241] [1 + 192] [289 \text{ mod } 254 + 1] [211 + 1] \\ &= [241] [193] [36] [212] \end{aligned}$$

Table 6 — Encoding ECI assignment numbers in ECC 200

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

5.4.2 ECIs and Structured Append

ECIs may occur anywhere in the message encoded in a single or Structured Append (see 5.6) 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.

5.4.3 Post-decode protocol

The protocol for transmitting ECI data shall be as defined in 11.4. When using ECIs, symbology identifiers (see 11.5) shall be fully implemented and the appropriate symbology identifier transmitted as a preamble.

5.5 ECC 200 symbol attributes

5.5.1 Symbol sizes and capacity

There are 24 square symbols and 6 rectangular symbols available in ECC 200. These are as specified in Table 7.

Table 7 — ECC 200 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 5.7.3

^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

5.5.2 Insertion of Alignment Patterns into larger symbols

As shown in Table 7, 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 are bounded by alignment patterns (see Annex D). The square symbols are divided into 4, 16, or 36 data regions (as illustrated in Annex D, Figures D.1, D.2, and D.3). The rectangular symbols are divided into two data regions (as illustrated in Annex D, 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.

5.6 Structured Append

5.6.1 Basic principles

Up to 16 ECC 200 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.

5.6.2 Symbol sequence indicator

This codeword indicates the position of the symbol within the set (up to 16) of ECC 200 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 8.

Table 8 — 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

EXAMPLE To indicate the 3rd symbol of a set of 7, this shall be encoded thus:

3rd position: 0010

Total 7 symbols: 1010

Bit pattern: 00101010

Codeword value: 42

5.6.3 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 64516 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.

5.6.4 FNC1 and Structured Append

If Structured Append is used in conjunction with FNC1 (see 5.2.4.6), 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.

5.6.5 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.

5.7 Error detection and correction

5.7.1 Reed-Solomon error correction

ECC 200 symbols employ Reed-Solomon error correction. For ECC 200 symbols with less than 255 total codewords, the error correction codewords are calculated from data codewords with no interleaving. For ECC 200 symbols with more than 255 total codewords, the error correction codewords are calculated from data codewords with the interleaving procedure described in Annex A. Each ECC 200 symbol has a specific number of data and error correction codewords which are divided into a specific number of blocks, as defined in Table 7, and to which the interleaving procedure defined in Annex A is applied.

The polynomial arithmetic for ECC 200 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 E.1.

5.7.2 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 E.1).

NOTE If this calculation is performed by "long division" the symbol data polynomial must 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 5. 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 ($n + 1 \dots n + k$ clock pulses), with the switch in the up position, the error correction codewords $\varepsilon_{k-1}, \dots, \varepsilon_0$ 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. If interleaving is used, the codewords will not be placed in consecutive symbol characters. (See Annex A).

Note: n and k are defined in 3.2 as the number of data codewords and the number of error correction codewords respectively.

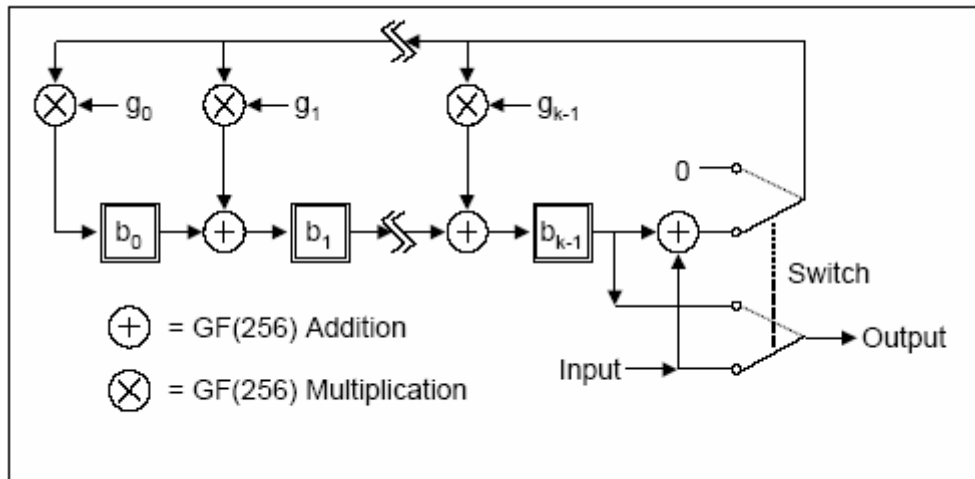


Figure 5 — Error correction codeword encoding circuit

5.7.3 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 the following formula:

$$e + 2t \leq d - p$$

where:

e = number of erasures

t = number of errors

d = number of error correction codewords

p = 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 $p = 1$).

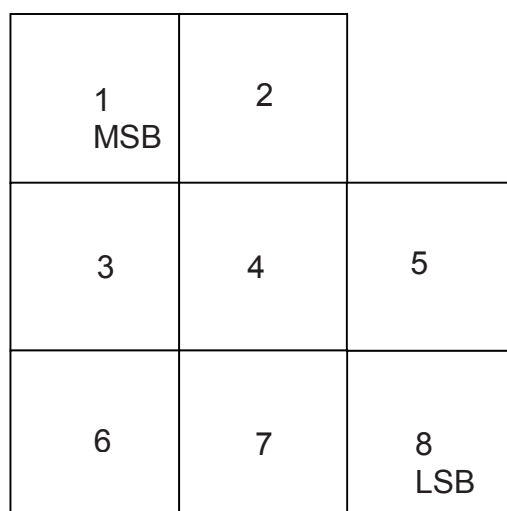
5.8 Symbol construction

Given the codeword sequence obtained in the previous sections, an ECC 200 symbol is constructed using the following steps:

- Place codeword modules in a mapping matrix
- Insert alignment pattern modules, if any
- Place finder modules along the perimeter

5.8.1 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 6. Because the symbol character shape defined in Figure 6 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 F.1, described in F.2 and illustrated in F.3.



LSB = Least significant bit
MSB = Most significant bit

Figure 6 — Representation of a codeword in a symbol character for ECC 200

5.8.2 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 7, 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 - D.3. For rectangular matrices, only a single vertical alignment pattern is placed between the data regions as shown in Figure D.4.

5.8.3 Finder Pattern module placement

Modules are placed along the perimeter of the matrix to construct the finder pattern as described in Section 4.3.1.

6 ECC 000 - 140 requirements

6.1 Use recommendations

For new applications or open systems ECC 200 is recommended (See Clause 5). There is no known application where ECC 200 will be more likely to succumb to symbol damage than ECC 000 to 140 for a given symbol size.

6.2 Encode procedure overview

This section provides an overview of the encoding procedure. Following sections will provide more details. An example encode for ECC 050 is given in Annex Q. The following steps convert user data to an ECC 000 - 140 symbol:

Step 1: Data encodation

The user data is analysed to identify the variety of different characters to be encoded. For maximum compaction efficiency, the lowest level encodation scheme capable of encoding the data should be selected. If the user does not specify the matrix size, then choose the smallest size that accommodates the data. The result of this step is called the Encoded Data Bit Stream.

Step 2: Data prefix construction

A Data Prefix Bit Stream is constructed from the Format ID, CRC, and Data Length bit fields. This Data Prefix Bit Stream is prefixed to the Encoded Data Bit Stream to produce the Unprotected Bit Stream.

Step 3: Error checking and correction

The Unprotected Bit Stream is processed by the user specified convolutional coding encode algorithm to produce the Protected Bit Stream. This step is omitted for ECC 000.

Step 4: Header and trailer construction

A header containing only the ECC bit field is prefixed to the Protected Bit Stream. A trailer containing pad bits (zeros) is appended to the Protected Bit Stream. The Protected Bit Stream with the header and trailer added is called the Unrandomised Bit Stream.

Step 5: Pattern randomising

The Unrandomised Bit Stream is processed by the pattern randomising algorithm and produces the Randomised Bit Stream.

Step 6: Module placement in matrix

Modules are placed in a matrix to construct the finder pattern. The Randomised Bit Stream is placed into the matrix one module at a time according to the data module placement algorithm given in Annex H. Figure 7 shows the various bit streams during the encode process.

6.3 Data encodation

The data shall be encoded using one of six encodation schemes (see Table 9). The encodation scheme is fixed for the entire symbol, and thus the selection of the most appropriate encodation scheme can have a considerable effect on the number of bits required to encode any given data. The same data may be represented in ECC 000 - 140 symbols in different ways through the use of the different encodation schemes. The character sets of all the encodation schemes, except the 8-bit byte scheme, are given in Annex I. The 8-bit byte scheme is user definable. The most efficient scheme to use is the lowest base number scheme

which is capable of encoding all the characters in the message. Thus if all the characters can be encoded in Base 27, it is not efficient to use Base 37, Base 41 or ASCII.

Table 9 — Encodation schemes

Encodation scheme	Characters	Bits per data character
Base 11	Numeric data	3,5
Base 27	Upper-case alphabetic	4,8
Base 37	Upper-case alphanumeric	5,25
Base 41	Upper-case alphanumeric and punctuation	5,5
ASCII	Full 128 ASCII set	7
8-bit Byte	User defined	8

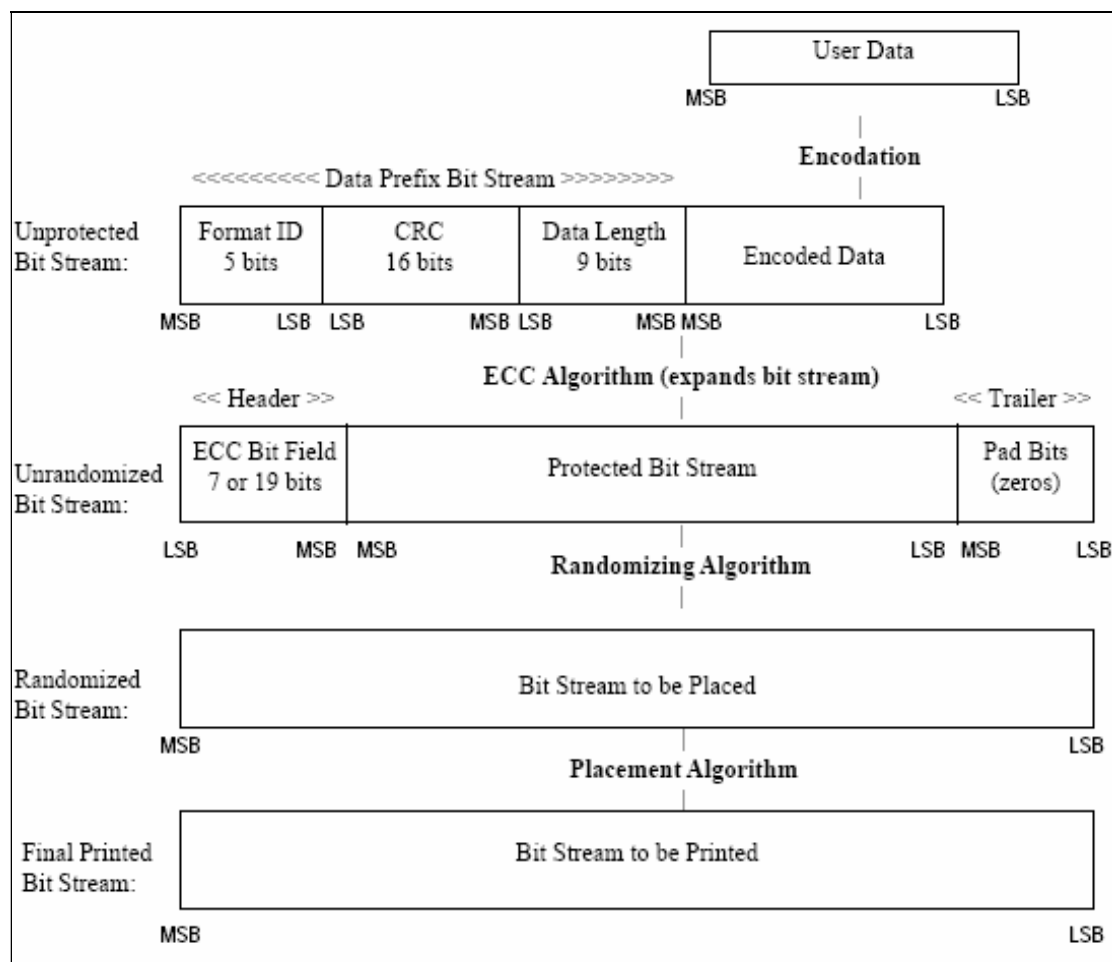


Figure 7 — ECC 000-140 encode process bit streams

To determine the appropriate encodation scheme, the data to be encoded should be analysed. The character sets of each of the Base *N* encodation schemes should be compared with the data character set to be encoded starting with the Base 11 character set. If this is suitable then it should be used, if not, the comparisons should continue with Base 27, Base 37 and Base 41, until the appropriate lowest level encodation scheme is found. If data characters beyond the capability of Base 41 need to be encoded, the ASCII set should be used, unless characters are beyond this; in which case the 8-bit byte set should be used.

For all encodation schemes, each compressed sequence of 4 to 24 bits is placed into the Encoded Bit Stream in reverse order (LSB first). This means that each individual compressed sequence is composed, then reversed, and output immediately to the Encoded Bit Stream. This does not mean that a complete compressed bit stream is formed, then reversed.

The details of each encodation scheme are given in the following clauses.

6.3.1 Base 11 - Numeric encodation

The Base 11 (Numeric) encodation scheme encodes 6 data characters as 21 bits, achieving an encodation density of 3,5 bits per data character. The Base 11 code set enables the following 11 characters to be encoded:

0 to 9

space

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 11 code values, as given in Annex I. In the second phase, the Base 11 code values shall be compacted using a Base 11 to Base 2 conversion according to the procedures defined in I.1.

6.3.2 Base 27 - Upper-case Alphabetic encodation

The Base 27 (Upper-case Alphabetic) encodation scheme encodes 5 data characters as 24 bits, achieving an encodation density of 4,8 bits per data character. The Base 27 code set enables the following 27 characters to be encoded:

A to Z

space

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 27 code values, as given in Annex I. In the second stage, the Base 27 code values shall be compacted using a Base 27 to Base 2 conversion according to the procedures defined in I.2.

6.3.3 Base 37 - Upper-case Alphanumeric encodation

The Base 37 (Upper-case Alphanumeric) encodation scheme encodes 4 data characters as 21 bits, achieving an encodation density of 5,25 bits per data character. The Base 37 code set enables the following 37 characters to be encoded:

A to Z

0 to 9

space

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 37 code values, as given in Annex I. In the second stage, the Base 37 code values shall be compacted using a Base 37 to Base 2 conversion according to the procedures defined in I.3.

6.3.4 Base 41 - Upper-case Alphanumeric plus Punctuation encodation

The Base 41 (Upper-case Alphanumeric plus Punctuation) encodation scheme encodes 4 data characters as 22 bits, achieving an encodation density of 5,5 bits per data character. The Base 41 code set enables the following 41 characters to be encoded:

A to Z

0 to 9

space

. (period)

, (comma)

- (minus or hyphen)

/ (forward slash or solidus)

The data is encoded in two stages. In the first stage, the actual data characters shall be replaced by their Base 41 code values, as given in Annex I. In the second stage, the Base 41 code values shall be compacted using a Base 41 to Base 2 conversion according to the procedures defined in I.4.

6.3.5 ASCII encodation

The ASCII encodation scheme enables all 128 characters from ISO/IEC 646 to be encoded. Each data character shall be encoded as a 7-bit byte equivalent to the decimal value shown in the ASCII column of Table I.1 of Annex I.

6.3.6 8-bit byte encodation

The 8-bit byte encodation scheme shall be used for closed applications, where the data interpretation shall be determined by the user. Each data character shall be encoded as an 8-bit byte.

6.4 User selection of error correction level

6.4.1 Selection of error correction level

ECC 000 - 140 symbols offer five levels of error correction using convolutional code error correction, as set out in Table 10. In an application, it is important to understand that these error correction levels result in the generation of a proportional increase in the number of bits in the message (and hence increase in the size of the symbol), and offer different levels of error recovery.

Table 10 — Error correction, error recovery and overhead percentages

Error correction code level	Maximum % damage	% increase in user bits from ECC 000
000	none	none
050	2,8	33
080	5,5	50
100	12,6	100
140	25	300

6.4.2 Other error correction levels based on convolutional code algorithms

Other levels of error correction, based on convolutional code algorithms, have been used in Data Matrix applications implemented prior to the publication of this International Standard. Information on these non-standard Error Correction levels is available from AIM Inc. Such symbols do not conform with this International Standard.

6.5 Constructing the Unprotected Bit Stream

Figure 7 illustrates that the Unprotected Bit Stream has the Data Prefix Bit Stream as a prefix to the encoded data bits. The component parts of the Data Prefix Bit Stream are defined below.

6.5.1 Format ID Bit Field

The format ID defines the data encodation scheme. The format ID has a decimal value for the purposes of definition and a 5-bit segment value for encoding as defined in Table 11.

Table 11 — Encoding the Format ID

Format ID	Encodation scheme	Binary segment value
		MSB LSB
1	Base 11	00000
2	Base 27	00001
3	Base 41	00010
4	Base 37	00011
5	ASCII	00100
6	8-bit Byte	00101

6.5.2 CRC Bit Field

The CRC Bit Field is generated by the CRC algorithm. The CRC Value is generated from the original user data as 8-bit bytes before encodation and so produces an independent error check on the user data. Annex J describes the complete procedure for generating the CRC Value.

6.5.3 Data Length Bit Field

The Data Length Bit Field is 9 bits in length and represents, as a binary value, the number of user data characters being encoded.

6.5.4 Data prefix construction

The Data Prefix Bit Stream is constructed as 30 bits as illustrated in Figure 8.

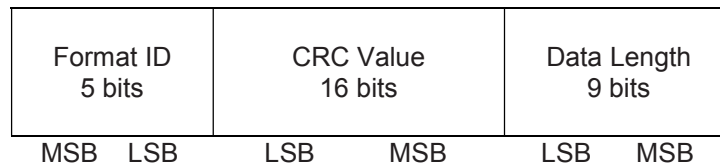


Figure 8 — Structure of Data Prefix Bit Stream

NOTE Some bit fields start with the MSB, others start with the LSB.

6.5.5 Completing the Unprotected Bit Stream

The encoded data bits are added as a suffix to the Data Prefix Bit Stream to construct the Unprotected Bit Stream.

6.6 Constructing the Unrandomised Bit Stream

Figure 7 illustrates that the Unrandomised Bit Stream has three constituent parts:

- a) Header
- b) Protected Bit Stream
- c) Trailer

The component parts shall be generated as defined below.

6.6.1 Header construction

The header of the Unrandomised Bit Stream contains the ECC Bit Field, which identifies the convolutional code structure used to protect the data encoded in the symbol. The ECC Bit Field is 7 or 19 bits long and the values are shown in Table 12.

Table 12 — ECC Bit Field

ECC Level	Binary Segment Identifier	
	MSB	LSB
000	1111110	
050	0001110000000001110	
080	1110001110000001110	
100	1111111110000001110	
140	1111110001110001110	

6.6.2 Applying convolutional coding to create the Protected Bit Stream

One of the five error correction levels shall be applied. The selection criteria are defined in Section 6.4. No error correction is applied for ECC 000, so the Unprotected Bit Stream becomes the Protected Bit Stream. For the other four error correction levels, convolutional coding is applied. This expands the user data proportionally throughout its length. The encoded bit stream shall be created by processing the unprotected bit stream through the appropriate error correction state machine and reading the results. The circuit diagrams of the four state machines for ECC 050 to 140 are given in Annex K.

6.6.3 Trailer construction

A Trailer containing pad bits (zeros) is appended to the Protected Bit Stream. Pad bits shall be added at the end of the bit stream to ensure that the square root of the total number of bits in the Unrandomised Bit Stream shall be an odd integer between 7 and 47. This ensures that the symbol is square.

6.6.4 Completing the Unrandomised Bit Stream

The Protected Bit Stream, with the header and trailer added, is called the Unrandomised Bit Stream and is shown in Figure 7.

6.7 Pattern randomising

The Unrandomised Bit Stream is processed by the pattern randomising algorithm and produces the Randomised Bit Stream. The pattern randomising algorithm consists of a bitwise XOR operation between the Unrandomised Bit Stream and the Master Random Bit Stream as given in Annex L starting with the MSB position and continuing for the length of the Unrandomised Bit Stream.

6.8 Module placement in matrix

The size of the sides of the data module grid is given by the odd integer square root (between 7 and 47) calculated in the procedure defined in 6.6.3. The Randomised Bit Stream is placed into the matrix one module at a time according to the data module placement grids given in Annex H. The finder pattern (as defined in 4.3.1) shall be placed to produce an external border to the data module grid.

7 Symbol dimensions

7.1 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 equal X .

alignment pattern: the width of the alignment pattern shall equal $2X$.

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

8 Symbol quality

Data Matrix symbols shall be assessed for quality using the 2D matrix bar code symbol print quality guidelines defined in ISO/IEC 15415, as augmented and modified below.

Some marking technologies may not be able to produce symbols conforming to this specification without taking special precautions. Annex T gives additional guidance to help any printing system achieve valid Data Matrix symbols.

8.1 Symbol quality parameters

8.1.1 Fixed pattern damage

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

NOTE As provided for in Annex A of ISO/IEC 15415, the measurements and values defined in Annex M of this International Standard override those indicated in Annex A of ISO/IEC 15415.

8.1.2 Scan grade and overall symbol grade

The scan grade shall be the lowest of the grades for symbol contrast, modulation, fixed pattern damage, decode, axial non-uniformity, grid non-uniformity and unused error correction in an individual image of the symbol. The overall symbol grade is the arithmetic mean of the individual scan grades for a number of tested images of the symbol.

8.1.3 Grid non-uniformity

The ideal grid is calculated by using the four corner points of the sampling grid for each data region and subdividing it equally in both axes.

8.1.4 Decode

The reference decode algorithm specified in this international standard 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.

8.2 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 are described in Annex R. These techniques do not constitute a print quality check of the produced symbols (the method specified earlier in this clause and Annex M is the required method for assessing symbol print quality) but they individually and collectively yield good indications of whether the symbol print process is creating workable symbols.

9 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. AC2

- AC₂** 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:
 - i) Follow the edge upward sampling pixel by pixel until either it reaches a point $3,5m_{min}$ 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 $3,5m_{min}$ 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 $3,5m_{min}$ from the starting point:
 - I) Plot a line A connecting the end points of the upward edge.
 - II) Test whether the intermediate edge points lie within $0,5m_{min}$ 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 $0,5m_{min}$ from line A. Back up to the closest edge point 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 $0,5m_{min}$ from line A. Back up to the closest edge point 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 $3,5m_{min}$ in step iii)IV), test if the downward edge reaches a point $3,5m_{min}$ 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 $2m_{min}$ from the starting point. If so, form an edge comprised of the appended $2m_{min}$ 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. **AC₂**

- AC₂ 3) Search among the saved line A1 segments for pairs of line segments that meet the following four criteria:
- 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} .
 - Verify that the two lines are co-linear within 5 degrees.
 - 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.
 - 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 $0,5m_{min}$ 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:
- Verify that the closest points on each line are separated by less than $1,5g_{max}$.
 - Verify that they are perpendicular within 5 degrees.
 - 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:
- Using a new horizontal scan line $3m_{min}$ 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.
 - Using a new vertical scan line $3m_{min}$ 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.
 - Repeat step 1) above except using a new horizontal scan line $3m_{min}$ below the centre horizontal scan line. If there is no decode, repeat step 2) above except using a new vertical scan line $3m_{min}$ right of the centre vertical scan line. If there is no decode, proceed to step 4) below. AC₂

- 4) Continue processing horizontal and vertical scan lines as in steps 1) through 3) that are $3m_{min}$ 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 9. 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 9.
 - 3) Move each search line away from the corner of the "L" as shown in Figure 9, 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 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. 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:

$$T = (\text{number of transitions}) (\text{"L" max. line length}) / (\text{search line length}).$$

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%.

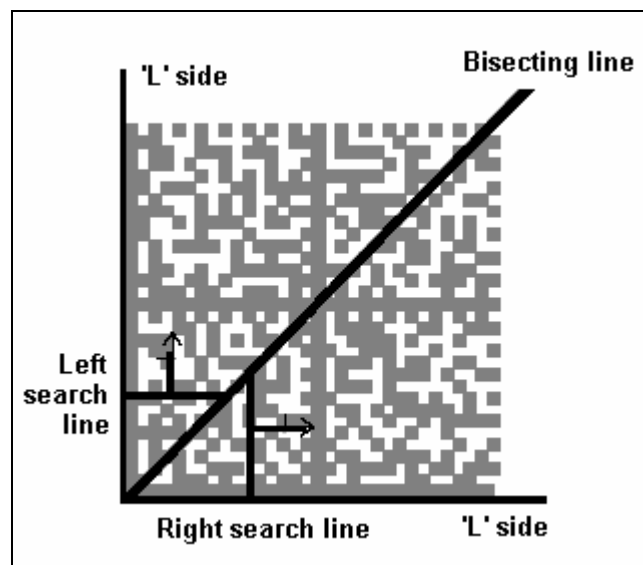


Figure 9 — Expanding search lines AC2

- 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 10.

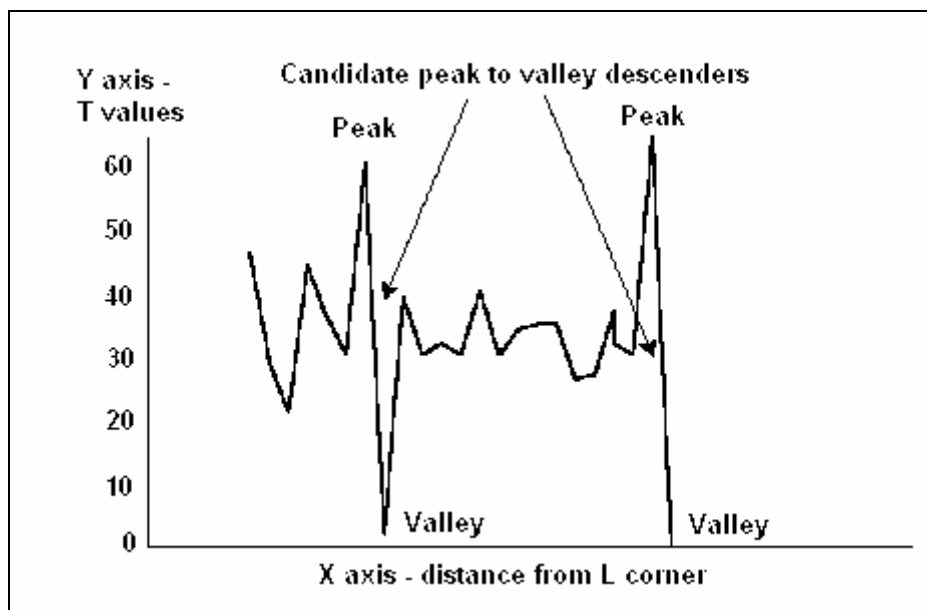


Figure 10 — Example plot of T as the search line expands

- 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 ($T_s = \text{maximum of zero and } T - 1$) 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 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 must 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:
- 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 11. AC2

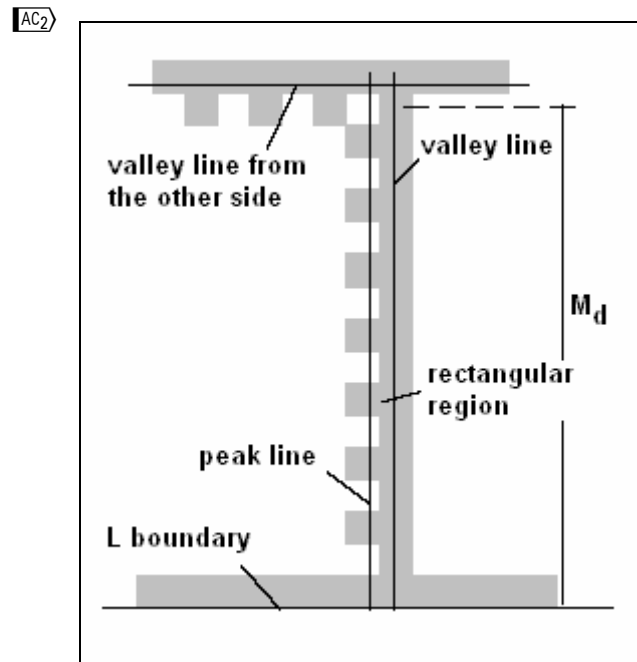


Figure 11 — Rectangular region construction

- 2) Within the rectangular region, find pixel edge pairs on the outside boundary of teeth:
 - i) Traverse test lines starting with and parallel to the valley line 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) If the number of transitions found is less than 15% of the number of pixels comprising the valley line, and the test line is not the peak line, move the test line toward the peak line by nominally one pixel and repeat step i), now considering new transitions in addition to those already found. If the 15% criterion is met or the peak line is reached, continue to the next step, otherwise continue searching from step d)6) for the next left peak and valley.
 - iii) Calculate a preliminary "best fit line" with linear regression using the points on the edge between the selected pixel pairs.
 - iv) Discard the 25% of the points which are furthest from the preliminary "best fit line". Calculate a final "best fit line" with linear regression using the remaining 75% of points. This line should pass along the outside of the alternating pattern, shown as the "best fit line" in Figure 12.
- 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:

$$\text{Offset} = \text{distance to the peak line} / ((\text{number of transitions} + 1) * 2)$$

Each of the two constructed lines should correspond to the mid-line of the alternating module pattern on that side, see Figure 12. AC2

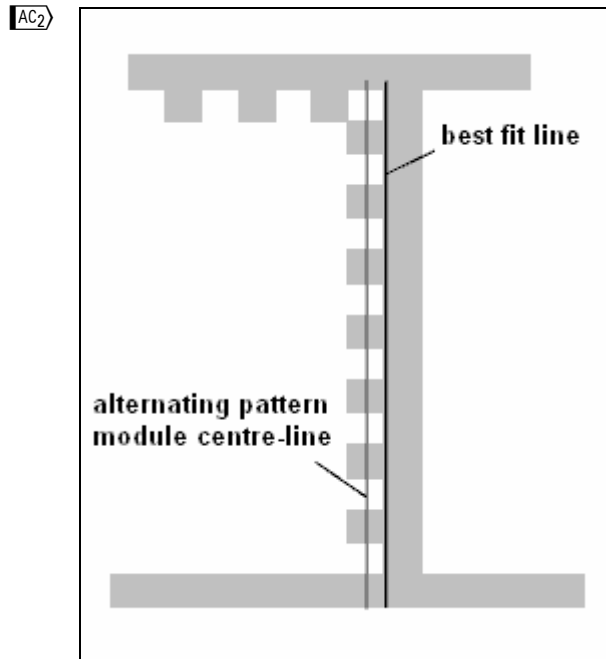


Figure 12 — Alternating pattern module centre-line

- f) For each side, measure the edge-to-edge distances in the alternating pattern:
- 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 11).
 - 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:

$$ink_spread = \text{Average} ((bar - ((bar + space) / 2)) / ((bar + space) / 2))$$

- 2) Calculate the centre of the bar in the median element pair using the following offset into the bar from the outside edge of the bar in the median pair:

$$offset = (EE_Dist * (1 + ink_spread)) / 4$$

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 are 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. AC2