INTERNATIONAL STANDARD

ISO/IEC 20830

First edition
2021-08

Information technology — Automatic identification and data capture techniques — Han Xin Code bar code symbology specification

Technologies de l'information — Techniques d'identification et de capture de données automatiques — Spécification des symboles du code à barres de Han Xin

Reference number
ISO/IEC 20830:2021(E)

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

Published in Switzerland

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 document 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).

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

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

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

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

Han Xin Code is a two-dimensional matrix symbology which is made up of an array of nominally square modules arranged in an overall square pattern, including a Finder Pattern located at four corners of the symbol that are intended to assist in easy locating of its position, size and inclination. Alignment Patterns and Assistant Alignment Patterns are also used in Versions 4 to 84 symbols. A wide range of size of symbols is provided together with four error correction levels. Module dimension is user-specified to produce symbols by a wide variety of techniques.

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. This document is published to meet this request.

Information technology — Automatic identification and data capture techniques — Han Xin Code bar code symbology specification

1 Scope

This document defines the requirements for the symbology known as Han Xin Code. It specifies the Han Xin Code symbology characteristics, data encoding process, symbol structure, dimensions and print quality requirements, error correction rules, reference decoding algorithm, and user-selectable application parameters.

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

ISO/IEC 15415:2011, *Information technology — Automatic identification and data capture techniques — Bar code symbol print quality test specification — Two-dimensional symbols*

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

ISO/IEC 15424, *Information technology — Automatic identification and data capture techniques — Data Carrier Identifiers (including Symbology Identifiers)*

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

GS1 General Specifications

3 Terms, definitions, and symbols

3.1 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/

3.1.1
assistant alignment pattern
non-data pattern located in the perimeter of the Han Xin Code symbol and intended to provide additional reference points to synchronize modules for reading

3.1.2
data bit stream
binary sequence that comprises the *information bit stream* (3.1.9) and the *error correction bit stream* (3.1.4)

3.1.3
data codeword
codeword that is used to encode information codewords and error correction codewords.

3.1.4
error correction bit stream
binary sequence used to correct errors, made by error correction encoding from the *information bit stream* (3.1.9)

3.1.5
Unicode
international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol applies across different platforms and programs.

Note 1 to entry: The repertoire of Unicode is synchronized with ISO/IEC 10646, and both are code-for-code identical.

3.1.6
GS1 mode
encoding *mode* (3.1.10) used for representing GS1 data in Han Xin Code

3.1.7
URI mode
encoding *mode* (3.1.11) used for representing uniform resource identifier (URI) described in RFC 3986 in Han Xin Code

3.1.8
Unicode mode
encoding *mode* (3.1.11) used for representing text data in *Unicode* (3.1.5) encoding/charset in Han Xin Code.

3.1.9
information bit stream
binary sequence made up of *mode* (3.1.11) encodings from the original input data

3.1.10
masking
XOR processing of the bit pattern in the information encoding region of the symbol with an algorithmically determined pattern to provide a symbol with more evenly balanced numbers of dark and light modules and to reduce the occurrence of patterns which would interfere with fast processing of the image

3.1.11
mode
method of representing a specific character set as a binary bit stream

3.1.12
mode indicator
bit sequence indicating in which *mode* (3.1.11) the following data sequence is encoded

3.1.13
mode terminator
bit sequence used to terminate the bit sequence representing an encoding *mode* (3.1.11)

3.1.14
padding bit
bit "0", appended to the *information bit stream* (3.1.9) to meet the requirements of the error correction algorithm

3.1.15
position detection pattern
one of the four pattern components of the Finder Pattern in Han Xin Code symbols

3.1.16
position detection center
center of the 3 × 3 dark modules in the *position detection pattern* (3.1.15)

3.1.17
position detection pattern separator
one-module wide non-data pattern, made up of all light modules, used to separate the *position detection pattern* (3.1.15) from the *structural information* (3.1.18) region

3.1.18
structural information
bit stream of data used to record *version* (3.1.20), error correction level and *masking* (3.1.10) solution

3.1.19
symbol padding bit
bit "0", not representing data, used to fill the empty positions of the symbol when the information encoding region cannot be fully filled with 8-bit codewords

3.1.20
version
size of the symbol represented in terms of its position in the sequence of permissible sizes for Han Xin Code symbols, from 23 × 23 modules (Version 1) to 189 × 189 modules (Version 84)

3.2 Mathematical and logical symbols

d	number of error correction codewords
e	number of erasures
k	total number of information codewords
n	total number of data codewords
t	number of errors
X	horizontal width of a module
Y	vertical distance from the center line of modules in one row to the center line of modules in an adjacent row
$(\ldots)_{bin}$	data in ( ) is figured in binary system
...HEX	data is figured in hexadecimal
$(\ldots)_{hex}$	data in ( ) is figured in hexadecimal

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

NOTE Without any specific statement, a byte usually comprises 8 binary bits and the byte's contents are represented in hexadecimal.

4 Symbology description

4.1 Symbology characteristics

4.1.1 Basic characteristics

4.1.1.1 General

Han Xin Code is a two-dimensional matrix symbology with the following basic characteristics:

4.1.1.2 Encodable characters

(1) Numeric characters (digits 0~9)

(2) ASCII characters (refer to ISO/IEC 646)

(3) Chinese characters (refer to GB18030)

(4) Octet bytes such as graphic and audio information, etc.

(5) GS1 data used in GS1 system

(6) Uniform Resource Identifier (URI)

(7) Any text data reference to a encoding/charset (such as Unicode, JIS, etc.)

4.1.1.3 Representation of data

A dark module is a binary one and a light module is a zero, Howerver, dark module is zero and a light module is one for the reflectance reversal symbols. See 4.1.2 for details of reflectance reversal.

4.1.1.4 Symbol size in modules

23 modules × 23 modules to 189 modules × 189 modules (Version 1 to 84, increasing in steps of two modules per side).

4.1.1.5 Maximum data capacity

(1) Numeric data: 7827 characters

(2) ASCII characters: 4350 characters

(3) Common Chinese Characters in Regions One and Two of GB18030: 2174 characters

(4) 2-byte Chinese characters data: 1739 characters

(5) 4-byte Chinese characters data: 1044 characters

(6) Binary byte data: 3261 bytes

4.1.1.6 Selectable error correction

Four levels of Reed-Solomon error correction and their error correction abilities of symbol codewords are shown as follows:

L1	8 %
L2	15 %
L3	23 %
L4	30 %

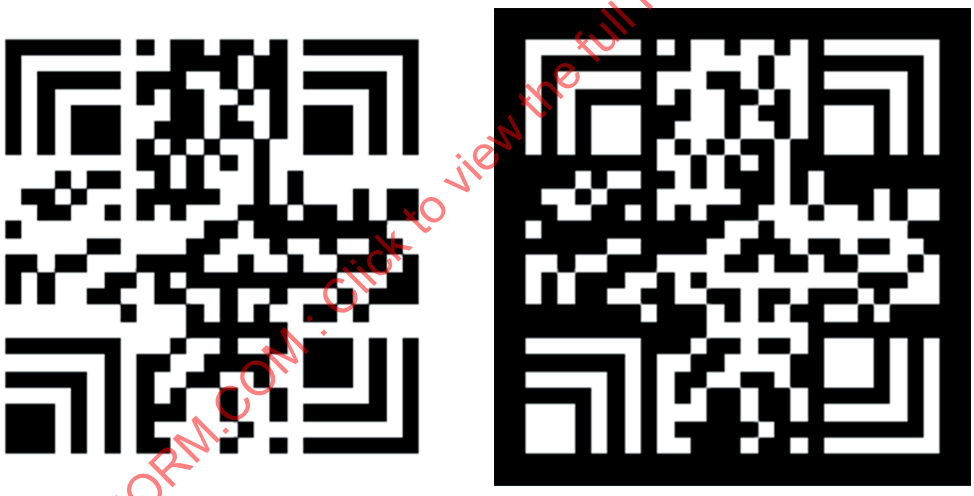
4.1.2 Summary of additional features

The following are additional inherent and optional features of Han Xin Code:

a. Masking: (Inherent) Masking pattern is used to the proportions of dark modules and light modules in the symbols and decrease the occurrences of images preventing fast processing.

b. 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).

c. Extended Channel Interpretations: (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.



a. Dark on light **b.** Light on dark

Figure 1 — Examples of Hanxin Code symbol encoding the text "Han Xin Code Symbol"

4.2 Symbol structure

4.2.1 General

Each Han Xin Code symbol shall be constructed of n × n nominally square modules set out in a regular square array and shall consist of an information encoding region, Structural Information regions and Fixed Pattern region. Fixed Pattern region include Finder Pattern, Position Detection Pattern separators, Alignment Pattern and Assistant Alignment Patterns. The symbol shall be surrounded by a quiet zone. Figure 2 illustrates a Version 24 Han Xin Code symbol. Figure 3 illustrates the structure of a Version 24 symbol.

Figure 2 — Han Xin Code symbol (Version 24)

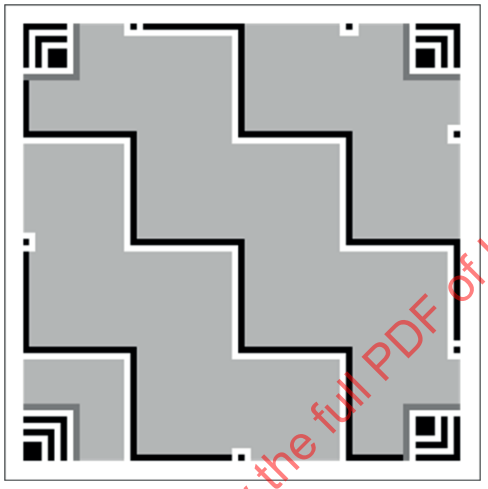


Figure 3 — Structure of a Version 24 Han Xin Code symbol

4.2.2 Symbol Versions and Sizes

There are eighty-four sizes of Han Xin Code symbol referred to as Version 1, Version 2 … Version 84 respectively. Version 1 measures 23 modules × 23 modules, Version 2 measures 25 modules × 25 modules and so on, increasing in steps of 2 modules per side up to Version 84 which measures 189 modules × 189 modules. Figure 4 to Figure 9 illustrate the symbols of Versions 1, 4, 24, 40, 62 and 84.

Figure 4 — Version 1 symbol

Figure 5 — Version 4 symbol

Figure 6 — Version 24 symbol

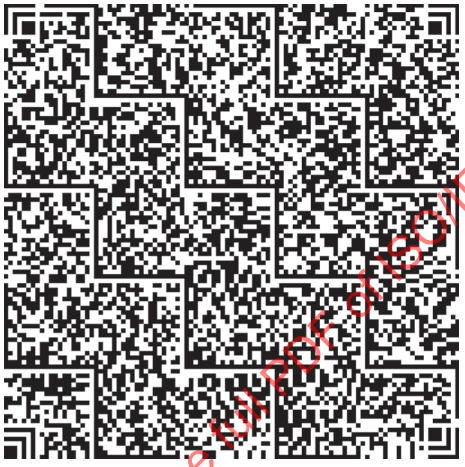


Figure 7 — Version 40 symbol

Figure 8 — Version 62 symbol

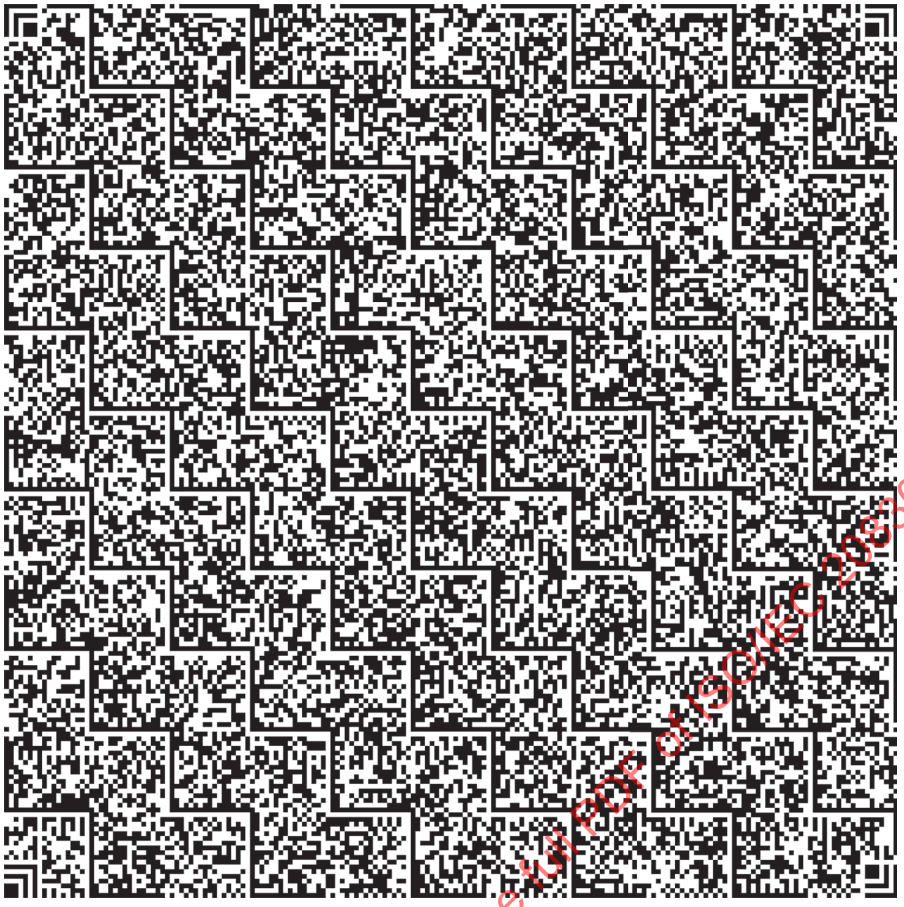


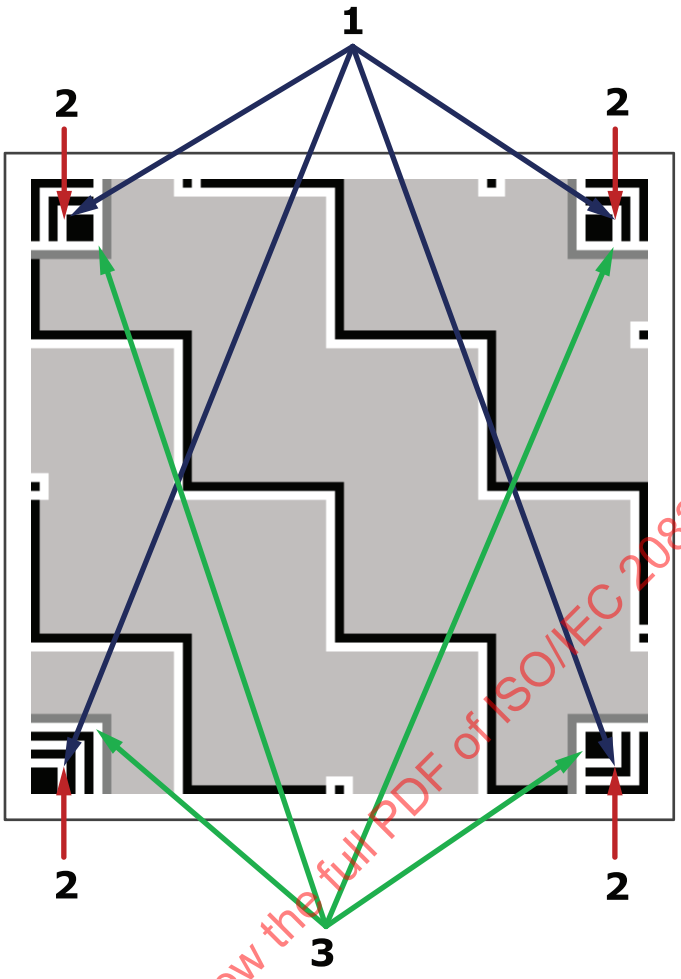
Figure 9 — Version 84 symbol

4.2.3 Finder Pattern

The Finder Pattern is made up of four Position Detection Patterns located at the four corners of the symbol respectively, which is illustrated in Figure 10.

The size of Position Detection Pattern is 7×7 modules. The Position Detection Pattern at the upper left corner is constructed by five aligned squares, whose lower right corners are superposed. The squares are constructed of dark 7 × 7 modules, light 6 × 6 modules, dark 5 × 5 modules, light 4 × 4 modules, dark 3 × 3 modules respectively. The other Position Detection Patterns are obtained by rotating the upper left one, as illustrated in Figure 10. The scanning ratio of each Position Detection Pattern is 1:1:1:1:3 or 3:1:1:1:1 (scanning along different directions) as illustrated in Figure 11. The center of dark 3 × 3 modules in Position Detection Pattern is called the Position Detection Center. Similar patterns have a low probability of being encountered elsewhere in the symbol, so that identification of the four Position Detection Patterns can define unambiguously the location and rotational orientation of the symbol in the field of view.

The shapes of all Position Detection Patterns are the same except their placement directions. Figure 11 is the Position Detection Pattern of the upper left corner. The lower left corner Position Detection Pattern for the upper left corner Position Detection Pattern by clockwise rotation of 90 degrees, the lower right corner Position Detection Pattern for the lower left corner Position Detection Pattern by clockwise rotation of 90 degrees, the upper right corner Position Detection Pattern for the lower right corner Position Detection Pattern by counterclockwise rotation of 90 degrees.



Key

1 Finder Pattern
2 Position Detection Pattern
3 Position Detection Pattern separator

Figure 10 — Finder Pattern of Han Xin Code symbol

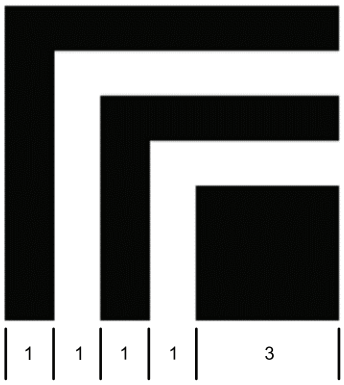


Figure 11 — Structure of Position Detection Pattern

4.2.4 Position Detection Pattern separator

A one-module wide separator, constructed of all light modules, is placed between each Position Detection Pattern and the Structural Information region, as illustrated in Figure 10.

4.2.5 Alignment Pattern

The Alignment Pattern is a set of step-wise alignment lines in the Han Xin Code symbol, as illustrated in Figure 12. There is no Alignment Pattern in the symbols of Version 1, Version 2, and Version 3.

The alignment line is made up of a dark line and an adjacent light line which are one module wide and follow the rule that the dark line is on the upside or right of the light line. On the left borderline and the bottom borderline of the symbol, the alignment line is a dark line in one module width.

There are two alignment lines in the Version 4 to 10 symbols with the length of k modules. When the version is bigger than 10, there are two kinds of Alignment Pattern layouts, the length of the two alignment lines on the lower left corner in the symbol is a special value r modules, the length of the rest regions is k modules except the alignment line on the upper left corner and the alignment line on the lower right corner is a special value (k-9) modules.. The relation of r, k and m in different versions is given by the following formula:

$$n = r + m \times k$$

where

n: is the total of modules in each side of the symbol;

r: is the selected length of the two specific alignment lines which locate near by the lower left corner of the Version 4 to 84 Han Xin code in module. *r* is the length of the dark line(including the turning module);

k: is the selected length of the normal alignment lines in module, which equals to the length of the light line(not including one of the turning modules)and the length of the dark line(not including one of the turning modules);

m: is a number obtained by dividing n minus r by *k*.

The relationship of r and k grids is shown in Figure 12.

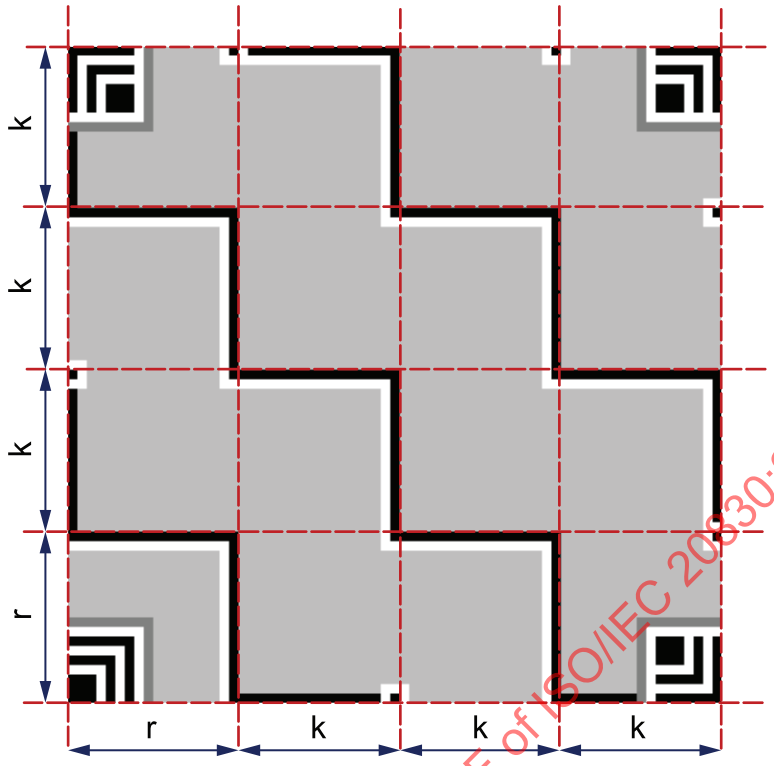
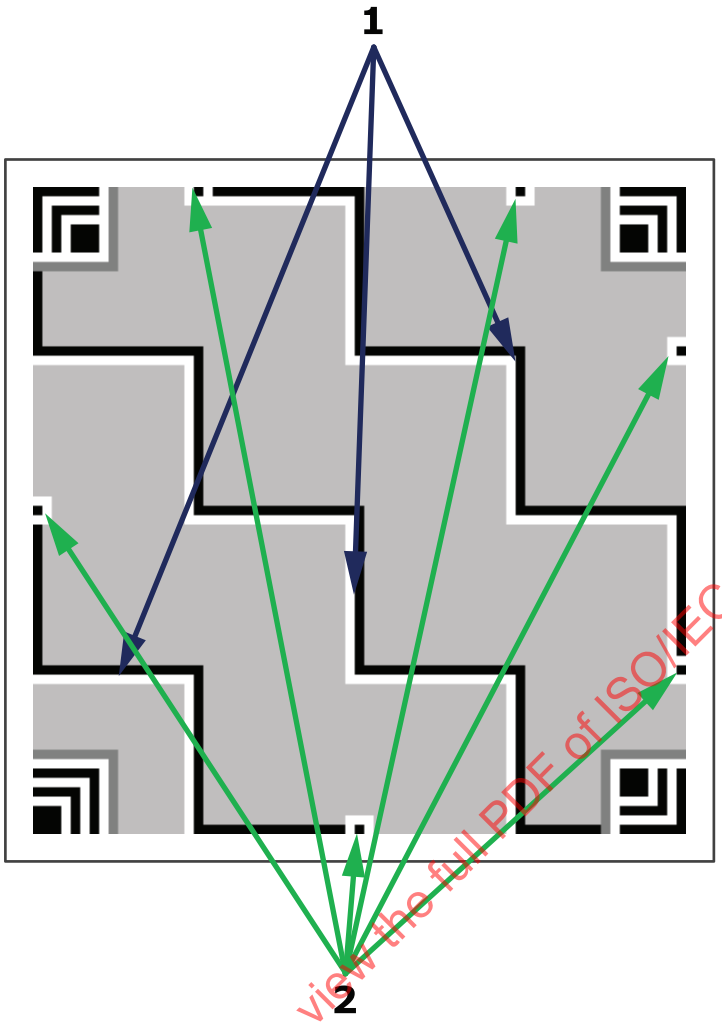


Figure 12 — Alignment Pattern of Version 24 Han Xin Code symbol

The parameters *n*, *k*, *m*, *r* in different versions of Han Xin Code shall conform to Table A.1 in Annex A.



Key

1 Alignment Pattern

2 Assistant Alignment Pattern

Figure 13 — Alignment Pattern and Assistant Alignment Pattern of Version 24 Han Xin Code symbol

4.2.6 Assistant Alignment Pattern

Assistant Alignment Pattern is made up of patterns comprised by 6 modules, including 5 light modules and a dark module, on the four sides of the symbol, as illustrated in Figure 13 and Figure 14.

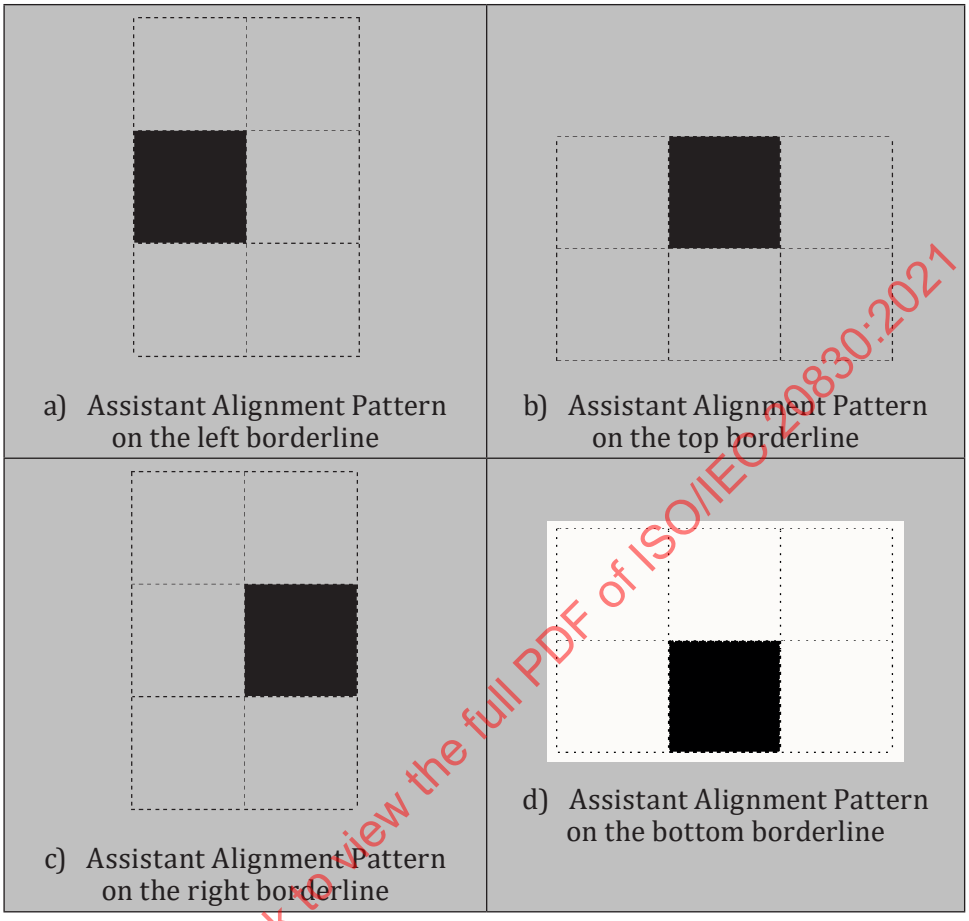
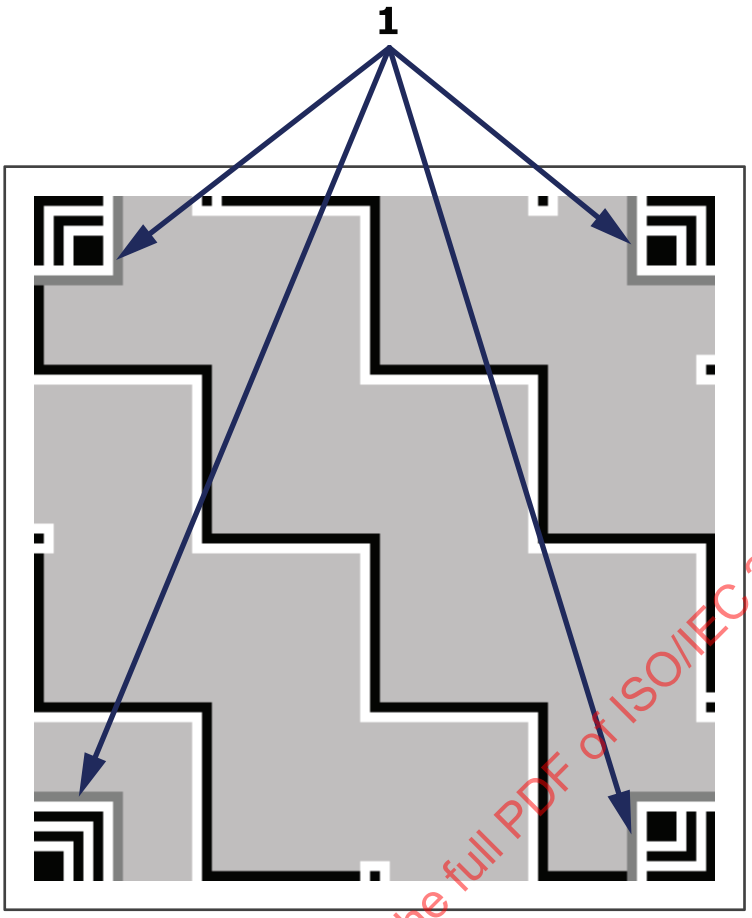


Figure 14 — Assistant Alignment Patterns

Every dark Alignment Pattern segment that points through data to a margin, without hitting another Fixed Pattern, aligns with the dark element of an Assistant Alignment Pattern, as illustrated in Figure 14.

4.2.7 Structural Information Region

Structural Information Region is a one module wide region surrounding the four Position Detection Pattern separator regions, beginning and ending on a margin, as illustrated in Figure 15. Every Structural Information region is made up of 17 modules, totaling 68 modules, which is used to encode Structural Information. See 5.8.3 for details.



Key

1 Structural Information Region

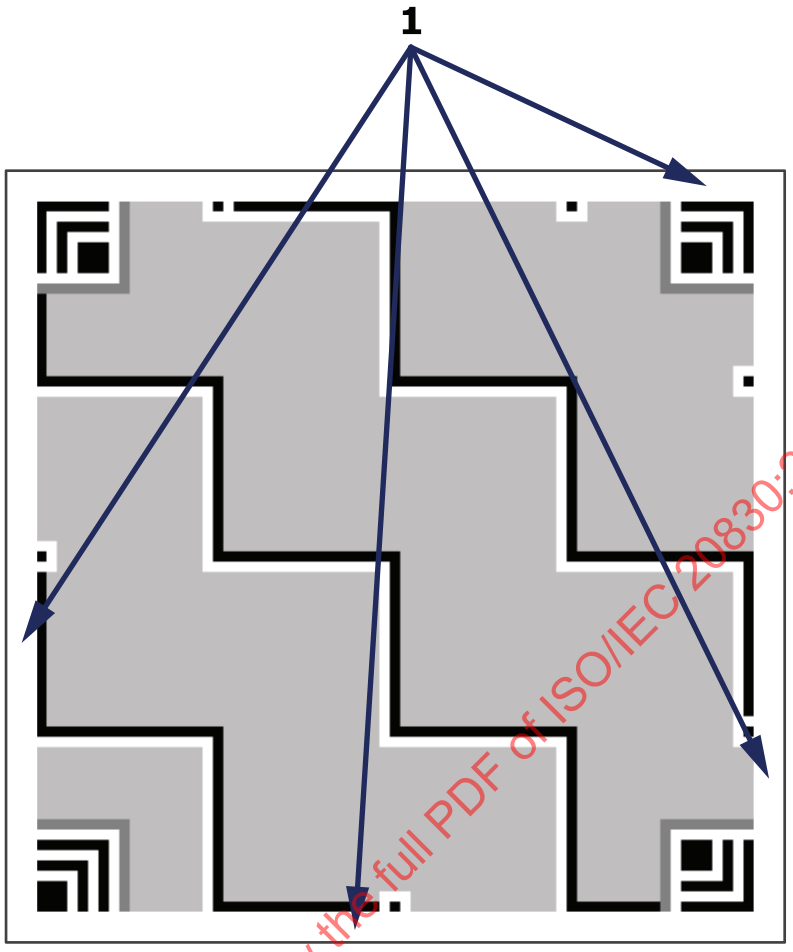
Figure 15 — Function information region of Han Xin Code symbol

4.2.8 Data Region

The data region encodes information codewords, padding bits, error correctloion codewords and symbol padding bits. It excludes the Fixed Patterns and the Structural Information region in the symbol.

4.2.9 Quiet Zone

This is a region which shall be at least three modules surrounding the symbol on all four sides. Nominal reflectance value of the Quiet Zone shall be equal to that of the light modules, as illustrated in Figure 16.



Key

1 Quiet Zone

Figure 16 — Quiet Zone

5 Requirements

5.1 Encode procedure overview

This subclause provides an overview of the encoding procedure.

The following steps are required to convert user data to a Han Xin Code symbol.

Step 1: Data analysis

The user data is analyzed to identify the variety of different characters to be encoded. Han Xin Code includes several modes (see 5.3) to allow different sub-sets of characters to be converted into symbol characters in efficient ways. Some additional data, such as mode indicators and mode terminators, are generated during mode transforming. These additional data should be fully considered when choosing modes in the process of input data analysis. the versions and data capacities of Han Xin Code symbols shall be identical with Annex B. In order to improve the compaction efficiency, the lowest level encoding scheme (the least bits per character) capable of encoding the data should be selected. When the encoding mode is decided, calculate the length of the data bit stream according to the length of information bit stream and error correction level. If the user does not specify the symbol version, then choose the smallest version that could accommodates the data.

Step 2: Data encoding

Convert the data characters into a bit stream in accordance with the rules for the mode in force, as defined in 5.3 and 5.4. Insert mode indicators as necessary to change modes at the beginning of each new mode segment, and mode terminators at the end of the data sequence. Split the resulting information bit stream into 8-bit codewords. Add pad bits as necessary to fill the number of data codewords required for the version and the error correction level.

Step 3: Error correction encoding

Divide the codewords sequence into the required number of blocks to enable the error correction algorithms to be processed. In each block, generate the error correction codewords, and append them at the end of the data codewords sequence.

Step 4: Constructing final data bit stream

In each error correction block, combine the information codewords sequence and error correction codewords sequence into a data codewords sequence. Construct the final data sequence by connecting all data codewords sequences in order.

Step 5: Module placement in matrix

Place the data modules in the matrix together with the final data bit stream, Finder Pattern, Position Detection Pattern separators, Alignment Pattern (if required), Assistant Alignment Pattern (if required) and symbol padding bits (if required).

Step 6: Data masking

Apply the data masking patterns in turn to the encoding region of the symbol. Evaluate the results and select the pattern which optimizes the dark and light module balance and minimizes the occurrence of undesirable patterns.

Step 7: Structural Information placement in matrix

Generate the Structural Information according to the version, error correction level and masking solution. Place the Structural Information in the matrix and complete the symbol.

5.2 Data analysis

If the user input data does not follow any Extended Channel Interpretation (ECI) protocol or use default ECI of Han Xin Code (\000003), analyze the user data to determine its content. Select the appropriate mode and divide the user data into sub-series of one or more encoding modes to encode each sequence as described in 5.3.

When the input data is GS1 data or URI, use GS1 mode or URI mode described in 5.3 to encode user data.

When the input data is numeric data, ASCII data and/or GB18030 characters etc., use numeric mode, Text mode, Chinese character modes and Binary Byte mode to encode the input data.

If the input data is in UTF8, and the input data cannot be efficiently encoded by other modes, the use Unicode mode to encode this data.

If user has not known or does not specify the encoding/charset/mode of input data, then analysis the input data, determine the lowest level encoding scheme (the least bits per character) capable of encoding the data (as described in 5.1 Step 2 and 5.4.15).

If the input data is known to be encoded differently than the default ECI of Han Xin Code (ECI 3), select the appropriate ECI mode to encode ECI data. The Extended Channel Interpretation (ECI) protocol allows the output data stream to have different interpretations from that of the default character set. The ECI protocol is defined consistently across a number of symbologies. Three broad types of interpretations are supported in Han Xin Code:

a. international character sets (or code pages)

b. general purpose interpretations such as encryption and compaction

c. user defined interpretations for closed systems

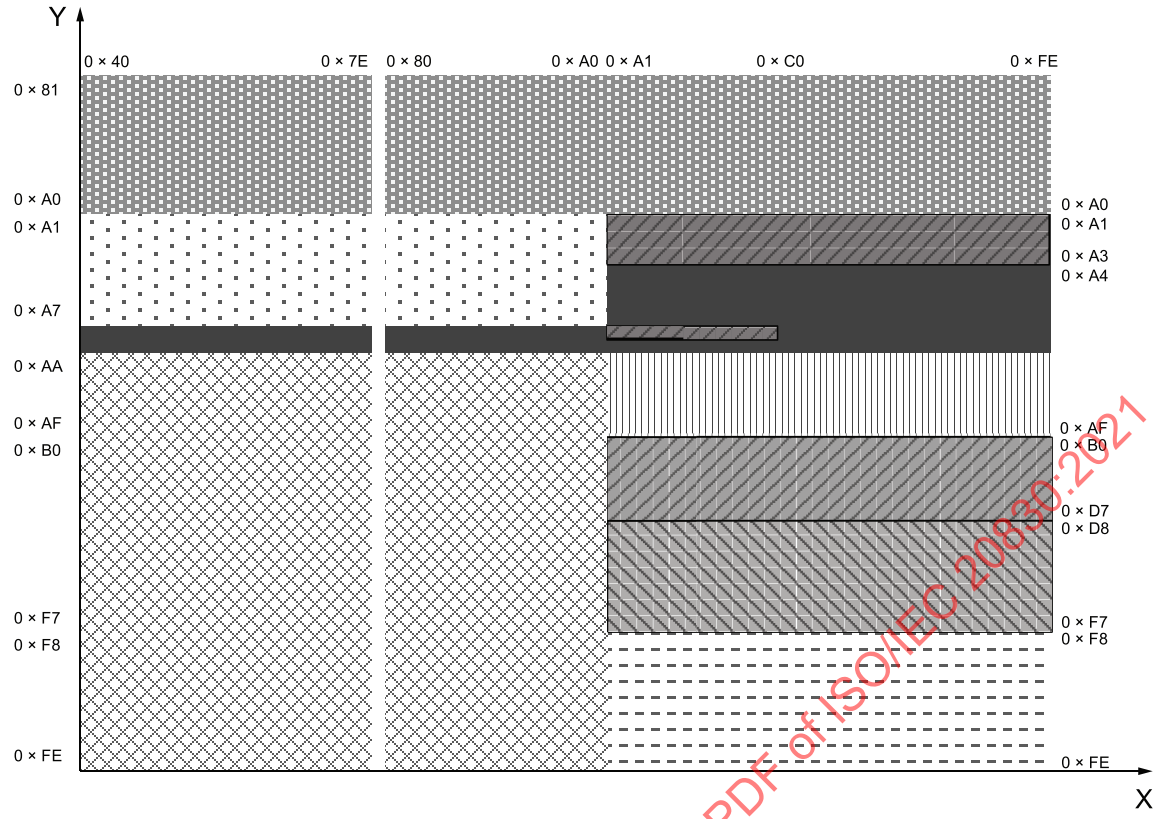
The Extended Channel Interpretation protocol is fully specified in the Extended Channel Interpretation (ECI) document. The protocol provides a consistent method to specify particular interpretations on byte values before printing and after decoding.

A specific Extended Channel Interpretation may be invoked anywhere in the encoded message. The default interpretation for Han Xin Code is ECI 000003 which shall represent the ISO/IEC 646 character set.

5.3 Mode

5.3.1 General

Han Xin Code has mode indicators which are used to define categories of the encoding data. The modes of Han Xin Code defined below are based on the character values and assignments associated with the default ECI. The data encoding modes for Han Xin Code are Numeric, Text mode, Binary Byte, Common Chinese Characters in Region One, Common Chinese Characters in Region Two, GB18030 2-byte region, GB18030 4-byte Region, ECI, URI, GS1, Unicode, etc. The mapping of Common Chinese Characters Mode One and Two are shown in Figure 17.



Key

X	First byte
Y	Second byte
	GB 18030 Region One
	GB 18030 Region Two
	GB 18030 Region Three
	GB 18030 Region Four
	GB 18030 User Defined Region One
	GB 18030 User Defined Region Two
	GB 18030 User Defined Region Three
	Han Xin Code Common Chinese Characters in Region One
	Han Xin Code Common Chinese Characters in Region Two

NOTE In GB18030, all the characters and symbols are designed to be encoded in one-byte, two-byte and four-byte, and the Chinese characters are encoded mainly in two bytes mode, which are separated to Region One and Region Two. The Chinese characters Mode One and Two are not equivalent to the Region One and Region Two defined in GB18030.

Figure 17 — Mapping of the Common Chinese Character Mode One and Two in GB18030

Each mode is indicated by specific mode indicator, as shown in Table 1.

Table 1 — Mode characteristics

Mode	Mode indicators	Bits per character
Reserved	$(0000)_{bin}$	
Numeric	$(0001)_{bin}$	3.3 (10 bits for three digits)
Text	$(0010)_{bin}$	6
Binary Byte	$(0011)_{bin}$	8
Common Chinese Characters in Region One	$(0100)_{bin}$	12
Common Chinese Characters in Region Two	$(0101)_{bin}$	12
GB18030 2-byte Region	$(0110)_{bin}$	15
GB18030 4-byte Region	$(0111)_{bin}$	21
ECI	$(1000)_{bin}$	N/A
Unicode	$(1001)_{bin}$	Adaptive
Reserved	$(1010)_{bin}$	
Reserved	$(1011)_{bin}$	
Reserved	$(1100)_{bin}$	
Reserved	$(1101)_{bin}$	
GS1	$(1110\ 0001)_{bin}$	N/A
URI	$(1110\ 0010)_{bin}$	N/A
Reserved	$(1111)_{bin}$	

See Annex C for Han Xin Code symbols of different versions and data capacities of different encoding modes.

5.3.2 Numeric mode

Numeric mode encodes data from decimal digit set (digits 0-9, byte values 30_{HEX} to 39_{HEX}). Normally, 3 data characters are represented by 10 bits

($(0000000000)_{bin}$ to $(1111100111)_{bin}$). The Numeric mode indicator is $(0001)_{bin}$.

5.3.3 Text mode

Text mode encodes data from common symbols defined in ISO/IEC 646, i.e. byte values 00_{HEX} to $1B_{HEX}$ and 20_{HEX} to $7F_{HEX}$. The Text mode is divided into two sub-modes, i.e. Text1 and Text2. The encoding range of Text1 is capital letters (A-Z), lowercase letters (a-z), and 10 numeric digits (0-9). The encoding range of text2 is the rest of the characters except ASCII 28 to 31 decimal. Every data character in Text mode is represented by 6 bits. The Text mode indicator is $(0010)_{bin}$.

5.3.4 Binary byte mode

Binary byte mode encodes binary data in any form and encodes them in their binary byte. Every byte in Binary Byte mode is represented by 8 bits. The Binary Byte mode indicator is $(0011)_{bin}$.

5.3.5 Common Chinese Characters in Region One mode

Every common Chinese Character in Region One is represented by 12 bits. Common Chinese Characters in Region one includes characters whose first byte value is in the range of $B0_{HEX}$ to $D7_{HEX}$ and second byte value is in the range of $A1_{HEX}$ to FE_{HEX} (3760 characters), and characters whose first byte value is in the range of $A1_{HEX}$ to $A3_{HEX}$, and second byte value is in the range of $A1_{HEX}$ to FE_{HEX} (282 characters), and characters whose byte values are in the range of $A8A1_{HEX}$ to $A8C0_{HEX}$ (32 characters). Common Chinese Characters in Region one are 4074 in total. The Common Chinese Characters in Region One mode indicator is $(0100)_{bin}$.

5.3.6 Common Chinese Characters in Region Two mode

Every common Chinese Character in Region Two is represented by 12 bits. Common Chinese Characters in Region Two which includes character whose first byte value is in the range of $D8_{HEX}$ to $F7_{HEX}$, and second byte value is in the range of $A1_{HEX}$ to FE_{HEX} (i.e., Chinese characters from 亍(chu) to 楚 (zha)). Common Chinese Characters in Region Two are 3008 in total. The common Chinese Characters in Region Two mode indicator is $(0101)_{bin}$.

5.3.7 GB18030 2-byte Region mode

Every GB18030 2-byte Region character is represented by 15 bits. The GB18030 2-byte Region encodes data from all characters (including the common Chinese Characters in Regions One and Two) in GB18030 double-byte region, (i.e., Chinese characters whose first byte value is in the range of 81_{HEX} to FE_{HEX} and second byte value is in the range of 40_{HEX} to $7E_{HEX}$ or 80_{HEX} to FE_{HEX}). GB18030 2-byte Region has 23940 Chinese characters in total. The GB18032 2-byte Region mode indicator is $(0110)_{bin}$.

5.3.8 GB18030 4-byte Region mode

Every GB18030 4-byte Region character is represented by 21 bits. GB18030 4-byte Region encodes data from all characters in GB18030 four-byte region (i.e., characters whose first byte value is in the range of 81_{HEX} to FE_{HEX}, and second byte value is in the range of 30_{HEX} to 39_{HEX}, and third byte value is in the range of 81_{HEX} to FE_{HEX}, and fourth byte value is in the range of 30_{HEX} to 39_{HEX}). The GB18030 4-byte Region has 1587600 characters in total. The GB18030 4-byte Region mode indicator is $(0111)_{bin}$.

5.3.9 Extended Channel Interpretation (ECI) mode

The Extended Channel Interpretation (ECI) mode indicates the encoded data is interpreted according to the ECI protocol defined by the AIM ECI Specifications. The ECI mode indicator is $(1000)_{bin}$. If all the data is in the default ECI 000003 there is no need to explicitly encode the ECI mode indicator $(1000)_{bin}$. Mode indicator $(1000)_{bin}$ is used to introduce ECI mode.

The Extended Channel Interpretation can only be used with readers enabled to transmit symbology identifiers. Readers that are not enabled to transmit the symbology identifier should not transmit the data from any symbol containing an ECI.

Apply one or more modes, such as numeric, Text mode, Common Chinese Character in Region One, etc., to encode its byte in maximum efficiency without considering its actual meaning. For example, a sequence of values in the range of 30_{HEX} to 39_{HEX} would be considered as a sequence of numbers (0 - 9) and encoded in numeric mode (see 5.3.2) even if they are not to be interpreted as numbers.

5.3.10 Unicode mode

Unicode mode designs a way to represent any text data reference to UTF8 encoding/charset in Han Xin Code. It is the intention of Han Xin Code to have Unicode-Mode-only symbols.The mode indicator of Unicode mode is $(1001)_{bin}$, the data analysis algorithm and encoding process are described in 5.4.12.

5.3.11 GS1 mode

GS1 mode indicates the data represented in Han Xin Code is GS1 data of GS1 system defined by GS1 General Specification. If Han Xin Code encodes GS1 data, the GS1 mode shall be entered as the first mode in the symbol. Other modes may be entered for data encodation efficiency but the transmission of the Symbology Identifier does not change, i.e., all the symbol data is GS1 data. GS1 Mode is not allowed to be used for non-GS1 data. If GS1 Mode is not the first mode and is entered somewhere other than the beginning of the symbol, the symbol is invalid.The mode indicator of GS1 mode is $(1110\ 0001)_{bin}$, the GS1 mode encoding algorithm is described in 5.4.13.

5.3.12 URI mode

Uniform Resource Identifier (URI) is widely used in bussiness and daily life of every person. URI mode indicates the data represented in Han Xin Code is Uniform Resource Identifier (URI) reference to RFC 3986.For most web based applications, the data encoded in a 2D barcode is treated as a URI, and there is no need to terminate the URI mode to start a new mode. It is the intention of this mode for a symbol to be used only for URI data. The mode indicator of URI mode is $(1110\ 0010)_{bin}$, the URI mode encoding algorithm is described in 5.4.14.

5.4 Data encoding

5.4.1 General

The encoding process consists of constructing information bit stream and constructing information codewords sequence.

5.4.2 Constructing the information bit stream

Select the mode required to encode any given data according to Numeric mode, Text mode, Common Chinese Characters in Region One mode, etc. (see 5.3.2 to 5.3.12).

The bit stream made by mode encoding should include:

a. Mode indicator (4 bits)

b. Character count indicator (for Binary Byte mode)

c. Mode bit stream after encoding

d. Mode terminator (for Numeric, Text, Common Chinese Characters in Region One, Chinese Characters in Region Two, GB18030 2-byte region, GB18030 4-byte region, Unicode mode, GS1 mode and URI mode).

Each mode segment shall begin with the first bit of the mode indicator and end with the final bit of the mode terminator. Construct the information bit stream by connecting all mode segments in order.

5.4.3 Constructing information codewords sequence

Determine the number of information codewords according to the error correction level and symbol version (see Annex B). Split the information bit stream into 8-bit codewords, and then add padding 0_{bin}-bits as necessary to fill the number of codewords required for the version and the error correction level.

5.4.4 Numeric mode encoding

In Numeric mode, the input data string is divided into groups of three digits (the number of digits in last group can be less than three), and each group is converted to its 10-bit binary equivalent (i.e. $(0000000000)_{bin}$–$(1111100111)_{bin}$). The binary data is then prefixed with the mode indicator and appended the mode terminator after the binary data sequence.

Table 2 specifies the mapping relationship about number of numeric characters in last group and mode terminators.

Table 2 — Mode terminators of Numeric mode

Number of numeric characters in last group	Mode terminator
1	$(1111111101)_{bin}$
2	$(1111111110)_{bin}$
3	$(1111111111)_{bin}$

EXAMPLE 1

Input data: 84613168549316542

1. Divide into groups of three digits: 846 131 685 493 165 42
2. Choose the Terminator:1111111110
3. Convert each group to its binary equivalent:

   846→1101001110

   131→0010000011

   685→1010101101

   493→0111101101

   165→0010100101

   42→0000101010
4. Connect the binary data in sequence:

1101001110 0010000011 1010101101 0111101101 0010100101 0000101010

5. Add the mode indicator and terminator to binary data:

0001 1101001110 0010000011 1010101101 0111101101 0010100101 0000101010 1111111110

EXAMPLE 2

Input data: 0019536472255

1. Divide into groups of three digits: 001 953 647 225 5
2. Choose the Terminator:1111111101
3. Convert each group to its binary equivalent:

   001→0000000001

   953→1110111001

   647→1010000111

   225→0011100001

   5→0000000101
4. Connect the binary data in sequence:

0000000001 1110111001 1010000111 0011100001 0000000101

5. Add the mode indicator and terminator to binary data:

0001 0000000001 1110111001 1010000111 0011100001 0000000101 1111111101

5.4.5 Text mode encoding

Text mode encodes data from a subset of the characters in ISO/IEC 646 (i.e., byte values 00_{HEX} – $1B_{HEX}$ and 20_{HEX} – $7F_{HEX}$). Each character is represented by 6 bits. The Text mode is divided into two sub-modes (i.e., Text1 and Text2). The encoding range of Text1 sub-mode is capital letters (A-Z),lowercase letters (a-z), and 10 numeric digits (0-9), as shown in Table 3. The encoding range of Text2 sub-mode is the rest of the symbols, as shown in Table 4. In addition, $(11110)_{bin}$ is defined as a transition between Text1 sub-mode and Text2 sub-mode. The mode terminator is $(111111)_{bin}$.

The default encoding sub-mode for Text mode is Text1 sub-mode. Characters to be encoded are read in order. If they are in Text1 sub-mode, they will be encoded according to the encoding values in Table 3; if they are in Text2 sub-mode, $(111110)_{bin}$ is first used to transfer mode from Text1 sub-mode to Text2 sub-mode, then the characters are encoded according to the encoding values in Table 4 until another sub-mode character is encountered or the character sequence is ended. $(111110)_{bin}$ is used to change back to Text1 sub-mode. The Text mode terminator $(111111)_{bin}$ is added followed by the generated encoding binary bit stream to end the mode's encoding.

Table 3 — Encoding/decoding mapping table of Text1 sub-mode

Character	ASCII value	Encoding value	Character	ASCII value	Encoding value
0	48	$(000000)_{bin}$	V	86	$(011111)_{bin}$
1	49	$(000001)_{bin}$	W	87	$(100000)_{bin}$
2	50	$(000010)_{bin}$	X	88	$(100001)_{bin}$
3	51	$(000011)_{bin}$	Y	89	$(100010)_{bin}$
4	52	$(000100)_{bin}$	Z	90	$(100011)_{bin}$
5	53	$(000101)_{bin}$	a	97	$(100100)_{bin}$
6	54	$(000110)_{bin}$	b	98	$(100101)_{bin}$
7	55	$(000111)_{bin}$	c	99	$(100110)_{bin}$
8	56	$(001000)_{bin}$	d	100	$(100111)_{bin}$
9	57	$(001001)_{bin}$	e	101	$(101000)_{bin}$
A	65	$(001010)_{bin}$	f	102	$(101001)_{bin}$
B	66	$(001011)_{bin}$	g	103	$(101010)_{bin}$
C	67	$(001100)_{bin}$	h	104	$(101011)_{bin}$
D	68	$(001101)_{bin}$	i	105	$(101100)_{bin}$
E	69	$(001110)_{bin}$	j	106	$(101101)_{bin}$
F	70	$(001111)_{bin}$	k	107	$(101110)_{bin}$
G	71	$(010000)_{bin}$	l	108	$(101111)_{bin}$
H	72	$(010001)_{bin}$	m	109	$(110000)_{bin}$
I	73	$(010010)_{bin}$	n	110	$(110001)_{bin}$
J	74	$(010011)_{bin}$	o	111	$(110010)_{bin}$
K	75	$(010100)_{bin}$	p	112	$(110011)_{bin}$
L	76	$(010101)_{bin}$	q	113	$(110100)_{bin}$
M	77	$(010110)_{bin}$	r	114	$(110101)_{bin}$
N	78	$(010111)_{bin}$	s	115	$(110110)_{bin}$
O	79	$(011000)_{bin}$	t	116	$(110111)_{bin}$
P	80	$(011001)_{bin}$	u	117	$(111000)_{bin}$
Q	81	$(011010)_{bin}$	v	118	$(111001)_{bin}$
R	82	$(011011)_{bin}$	w	119	$(111010)_{bin}$
S	83	$(011100)_{bin}$	x	120	$(111011)_{bin}$
T	84	$(011101)_{bin}$	y	121	$(111100)_{bin}$

Table 3 *(continued)*

Character	ASCII value	Encoding value	Character	ASCII value	Encoding value
U	85	$(011110)_{bin}$	z	122	$(111101)_{bin}$

Table 4 — Encoding/decoding mapping table of Text2 sub-mode

Character	ASCII value	Encoding value	Character	ASCII value	Encoding value
NUL	0	$(000000)_{bin}$	#	35	$(011111)_{bin}$
SOH	1	$(000001)_{bin}$	$	36	$(100000)_{bin}$
STX	2	$(000010)_{bin}$	%	37	$(100001)_{bin}$
ETX	3	$(000011)_{bin}$	&	38	$(100010)_{bin}$
EOT	4	$(000100)_{bin}$	‘	39	$(100011)_{bin}$
ENQ	5	$(000101)_{bin}$	(	40	$(100100)_{bin}$
ACK	6	$(000110)_{bin}$	)	41	$(100101)_{bin}$
BEL	7	$(000111)_{bin}$	*	42	$(100110)_{bin}$
BS	8	$(001000)_{bin}$	+	43	$(100111)_{bin}$
HT	9	$(001001)_{bin}$	,	44	$(101000)_{bin}$
LF	10	$(001010)_{bin}$	-	45	$(101001)_{bin}$
VT	11	$(001011)_{bin}$	.	46	$(101010)_{bin}$
FF	12	$(001100)_{bin}$	/	47	$(101011)_{bin}$
CR	13	$(001101)_{bin}$	:	58	$(101100)_{bin}$
SO	14	$(001110)_{bin}$	;	59	$(101101)_{bin}$
SI	15	$(001111)_{bin}$	<	60	$(101110)_{bin}$
DLE	16	$(010000)_{bin}$	=	61	$(101111)_{bin}$
DC1	17	$(010001)_{bin}$	>	62	$(110000)_{bin}$
DC2	18	$(010010)_{bin}$	?	63	$(110001)_{bin}$
DC3	19	$(010011)_{bin}$	@	64	$(110010)_{bin}$
DC4	20	$(010100)_{bin}$	[	91	$(110011)_{bin}$
NAK	21	$(010101)_{bin}$	\	92	$(110100)_{bin}$
SYN	22	$(010110)_{bin}$	]	93	$(110101)_{bin}$
ETB	23	$(010111)_{bin}$	^	94	$(110110)_{bin}$
CAN	24	$(011000)_{bin}$	_	95	$(110111)_{bin}$
EM	25	$(011001)_{bin}$	`	96	$(111000)_{bin}$
SUB	26	$(011010)_{bin}$	{	123	$(111001)_{bin}$
ESC	27	$(011011)_{bin}$	|	124	$(111010)_{bin}$
SP	32	$(011100)_{bin}$	}	125	$(111011)_{bin}$
!	33	$(011101)_{bin}$	~	126	$(111100)_{bin}$
”	34	$(011110)_{bin}$	DEL	127	$(111101)_{bin}$

5.4.6 Binary mode encoding

Binary Byte mode is used to encode binary data in any form. The linking of Binary Byte mode indicator, count indicator and binary byte sequence form an information bit stream of binary byte encoding mode. The count indicator is a 13-bit binary sequence and is represented as the byte number of input binary data.

5.4.7 Common Chinese Characters in Region One mode encoding

Common Chinese Characters in Region One mode encodes common characters and symbols in double-byte region one and two of GB18030, 4074 characters in total. Each character or symbol is represented by 12-bit binary sequence according to the following method. The encoding value of the character in this mode is in the range of $(000000000000)_{bin}$ to $(111111101001)_{bin}$.

a. For characters and symbols with GB18030 values whose first byte value is in the range of $B0_{HEX}$ to $D7_{HEX}$ and second byte value is in the range of $A1_{HEX}$ to FE_{HEX}:

   (1) Subtract $A1_{HEX}$ from the second byte value.

   (2) Subtract $B0_{HEX}$ from the first byte value.

   (3) Multiply the result of (2) by $5E_{HEX}$.

   (4) Add the result of (2) to the product from (3).

   (5) Convert the result to a 12-bit binary string.

b. For characters and symbols with GB18030 values whose first byte value is in the range of $A1_{HEX}$ to $A3_{HEX}$ and second byte value is in the range of $A1_{HEX}$ to FE_{HEX}:

   (1) Subtract $A1_{HEX}$ from the second byte value.

   (2) Subtract $A1_{HEX}$ from the first byte value.

   (3) Multiply the result of (2) by $5E_{HEX}$.

   (4) Add the result of (1) and $EB0_{HEX}$ to the product from (3).

   (5) Convert the result to a 12-bit binary string.

c. For characters and symbols with GB18030 values from $A8A1_{HEX}$ to $A8C0_{HEX}$, subtract $A1_{HEX}$ from the second byte value then add FCA_{HEX} to the result, then convert the result into a 12-bit binary string.

The information bit stream is generated in the following manner. Begin with mode indicator $(0100)_{bin}$. Encode characters in order into 12-bit binary and connect them into a sequence. Add the mode terminator $(111111111111)_{bin}$. Additionally, $(111111111110)_{bin}$ is considered as transition from Common Character In Region One to Common Character In Region Two.

EXAMPLE 1

Input character : 全

Character value:	$C8AB_{HEX}$
(1) Subtract $B0_{HEX}$ from its first byte value	$C8_{HEX}$-$B0_{HEX}$=18_{HEX}
(2) Multiply the result of (1) by $5E_{HEX}$	18_{HEX}×$5E_{HEX}$=$8D0_{HEX}$
(3) Subtract $A1_{HEX}$ from its second byte value	AB_{HEX}-$A1_{HEX}$=$0A_{HEX}$
(4) Add the result of (2) to result of (3)	$8D0_{HEX}$+$0A_{HEX}$=$8DA_{HEX}$
(5) Convert the result to a 12-bit binary string	$(100011011010)_{bin}$

EXAMPLE 2

Input character: 

Character value:	$A3BB_{HEX}$
(1) Subtract $A1_{HEX}$ from its first byte value	$A3_{HEX}$-$A1_{HEX}$=02_{HEX}
(2) Multiply the result of (1) by $5E_{HEX}$	02_{HEX}×$5E_{HEX}$=BC_{HEX}
(3) Subtract $A1_{HEX}$ from its second byte	BB_{HEX}-$A1_{HEX}$=$1A_{HEX}$
(4) Add the result of (2), result of (3) and $EB0_{HEX}$	BC_{HEX}+$1A_{HEX}$+$EB0_{HEX}$=$F86_{HEX}$
(5) Convert the result to a 12-bit binary string	$(1111\ 1000\ 0110)_{bin}$

EXAMPLE 3

Input character: 

Character value:	$A8BE_{HEX}$
(1) Subtract $A1_{HEX}$ from its second byte value	BE_{HEX} -$A1_{HEX}$=$1D_{HEX}$
(2) Add the result of (1) and FCA_{HEX}	$1D_{HEX}$+FCA_{HEX}=$FE7_{HEX}$
(3) Convert the result to a 12-bit binary string	$(111111100111)_{bin}$

5.4.8 Common Chinese Characters in Region Two mode encoding

Common Chinese Characters in Region Two mode encodes common characters and symbols in double-byte region two of GB18030, 3008 characters in total. Each character or symbol is represented by 12-bit binary sequence according to following method. The encoding value of the character in this mode is in the range of $(000000000000)_{bin}$ to $(101110111111)_{bin}$.

For characters and symbols with GB18030 values whose first byte value is in the range of $D8_{HEX}$ to $F7_{HEX}$ and second byte value is in the range of $A1_{HEX}$ to FE_{HEX}:

(1) Subtract $A1_{HEX}$ from the second byte value.

(2) Subtract $D8_{HEX}$ from the first byte value.

(3) Multiply the result of (2) by $5E_{HEX}$.

(4) Add the result of (1) to the product from (3).

(5) Convert the result to a 12-bit binary string.

The information bit stream is generated in the following manner. Begin with mode indicator $(0101)_{bin.}$ Encode characters in order into 12-bit binary and connect them into a sequence. Add the mode terminator $(111111111111)_{bin}$. Additionally, $(111111111110)_{bin}$ is considered as transition from Common Character in Region Two to Common Character in Region One.

EXAMPLE

Input character : 

Character value:	$F3A3_{HEX}$
(1) Subtract $D8_{HEX}$ from its first byte value	$F3_{HEX}$-$D8_{HEX}$=$1B_{HEX}$
(2) Multiply the result of (1) by $5E_{HEX}$	$1B_{HEX}$×$5E_{HEX}$=$9EA_{HEX}$
(3) Subtract $A1_{HEX}$ from its second byte value	$A3_{HEX}$-$A1_{HEX}$=02_{HEX}
(4) Add the result of (3) to the product from (2)	$9EA_{HEX}$+02_{HEX}=$9EC_{HEX}$
(5) Convert the result to a 12-bit binary string	$(100111101100)_{bin}$

5.4.9 GB18030 2-byte Region mode encoding

GB18030 2-byte Region mode encodes all characters and symbols in double-byte region of GB18030, i.e. Chinese characters whose first byte value is in the range of 81_{HEX} to FE_{HEX} and second byte value is in the range of 40_{HEX} to $7E_{HEX}$ or in the range of 80_{HEX} to FE_{HEX}, 23940 Chinese characters in total. Characters and symbols in this mode are represented by 15-bit binary sequence according to following method. The encoding value of the character in this mode is in the range of $(000000000000000)_{bin}$ to $(101110110000011)_{bin}$.

The following steps are the complete 2-bit Byte Chinese Characters Mode Encoding process:

(1) For characters whose second byte value is in the range of 40_{HEX} to $7E_{HEX}$, subtract 40_{HEX} from its second byte value. For characters whose second byte value is in the range of 80_{HEX} to FE_{HEX}, subtract 41_{HEX} from its second byte value.

(2) Subtract 81_{HEX} from the first byte value.

(3) Multiply the result of (2) by BE_{HEX}.

(4) Add the result of (1) to the product from (3).

(5) Convert the result to a 15-bit binary string.

The information bit stream is generated in the following manner. Begin with mode indicator $(0110)_{bin}$. Encode characters in order into 15-bit binary and connect them into a sequence. Add the mode terminator $(111111111111111)_{bin}$.

EXAMPLE 1

Input character : 

Character value:	$9D51_{HEX}$
(1) Subtract 81_{HEX} from its first byte value	$9D_{HEX}$ -81_{HEX} =$1C_{HEX}$
(2) Multiply the result of (1) by BE_{HEX}	$1C_{HEX}$×BE_{HEX} =$14C8_{HEX}$
(3) Subtract 40_{HEX} from its second byte value	51_{HEX}-40_{HEX}=11_{HEX}
(4) Add the result of (3) to the product from (2)	$14C8_{HEX}$ +11_{HEX} =$14D9_{HEX}$
(5) Convert the result to a 15-bit binary sequence	$(001010011011001)_{bin}$

EXAMPLE 2

Input character : 

Character value: $9EAF_{HEX}$

(1) Subtract 81_{HEX} from its first byte value $9E_{HEX}$-81_{HEX}=$1D_{HEX}$

(2) Multiply the result of (1) by BE_{HEX} $1D_{HEX}$×BE_{HEX}=1586_{HEX}

(3) Subtract 41_{HEX} from its second byte AF_{HEX}-41_{HEX}=$6E_{HEX}$

(4) Add the result of (3) to the product from (2) 1586_{HEX}+$6E_{HEX}$=$15F4_{HEX}$

(5) Convert the result to a 15-bit binary sequence$(001010111110100)_{bin}$

5.4.10 GB18030 4-byte Region mode encoding

GB18030 4-byte Region mode encodes all characters and symbols in four-byte region of GB18030, i.e. Chinese characters and symbols whose first byte value is in the range of 81 $_{HEX}$ to FE $_{HEX}$, second byte value is in the range of 30 $_{HEX}$ to 39 $_{HEX}$, third byte value is in the range of 81 $_{HEX}$ to FE_{HEX} and fourth byte value is in the range of 30 $_{HEX}$ to 39 $_{HEX}$, 1,587,600 Chinese characters and symbols in total. This mode does not have a mode terminator. By default, this encoding mode will be ended automatically after encoding a character. Each of the characters and symbols are represented by a 21-bit binary sequence with a leading four-bit mode indicator $(0111)_{bin}$ according to the following method.

The encoding value of the character in this mode is in the range of $(000000000000000000000)_{bin}$ to $(110000011100110001111)_{bin}$. A four-bit mode indicator is added in front of the sequence.

The following steps are encoding method:

(1) Subtract 30_{HEX} from the fourth byte value.

(2) Subtract 81_{HEX} from the third byte value.

(3) Subtract 30_{HEX} from the second byte value.

(4) Subtract 81_{HEX} from the first byte value.

(5) Multiply the result of (2) by $0A_{HEX}$.

(6) Multiply the result of (3) by $04EC_{HEX}$.

(7) Multiply the result of (4) by 3138_{HEX}.

(8) Add the result of (1), the product from (5), the product from (6) and the product from (7) together.

(9) convert the result to a 21-bit binary sequence.

EXAMPLE

Input character : 

Character value: $8139EF30_{HEX}$

(1) Subtract 81_{HEX} from its first byte value 81_{HEX}-81_{HEX}=00_{HEX}

(2) Multiply the result of (1) by 3138 $_{HEX}$ 00 $_{HEX}$×3138_{HEX}=00 $_{HEX}$

(3) Subtract 30 $_{HEX}$ from its second byte value 39_{HEX}-30_{HEX}=09_{HEX}

(4) Multiply the result of (3) by 04EC $_{HEX}$ $09_{HEX} \times 04EC_{HEX} = 2C4C_{HEX}$

(5) Subtract 81_{HEX} from its third byte value $EF_{HEX} - 81_{HEX} = 6E_{HEX}$

(6) Multiply the result of (5) by OA_{HEX} $6E_{HEX} \times 0A_{HEX} = 44C_{HEX}$

(7) Subtract 30_{HEX} from the fourth byte value $30_{HEX} - 30_{HEX} = 00_{HEX}$

(8) Add the results of (2), (4), (6) and (7) together

$$00_{HEX} + 2C4C_{HEX} + 44C_{HEX} + 00_{HEX} = 3098_{HEX}$$

(9) Convert the result to a 21-bit binary sequence $(000000011000010011000)_{bin}$

5.4.11 ECI mode encoding

5.4.11.1 General

ECI mode is invoked by the use of mode indicator $(1000)_{bin}$. The data sequence is encoded according to the rules of Numeric mode, Text mode, etc. See encoding rules described in 5.4. Each mode segment begins with the highest bit of mode indicator and ends with the lowest bit of data bit stream (for Binary Byte mode) or mode terminator (for Numeric mode, Text mode, Common Chinese Characters in Region One mode, Common Chinese Characters in Region Two mode, GB18030 2-byte Region mode, GB18030 4-byte Region mode). The length of each mode segment is determined definitely by the encoding mode rules; there is no separator between segments. The ECI encoding sequence is generated by connecting the ECI designator, ECI assignment numbers and one or more binary sequences encoded by related ECI mode.

5.4.11.2 ECI designator

The ECI designator appears in the data to be encoded as character $5C_{HEX}$ ("\" or the backslash in ISO/IEC 646) followed by the six digit assignment number. Where $5C_{HEX}$ appears as true data it shall be doubled in the data string before encoding in symbols to which the ECI protocol applies. Each ECI is designated by a six-digit assignment number which is encoded in the Han Xin Code symbol as the one, two or three bytes following the ECI mode indicator. The encoding rules are defined in Table 5. The lower numbered ECI assignments may be encoded in multiple ways, but the shortest way is preferred.

Table 5 — Encoding ECI assignment numbers

ECI assignment value	Number of encoding bit	Codeword values
000000 to 000127	8	0bbbbbbb
000000 to 016383	16	10bbbbbb bbbbbbbb
000000 to 811799	24	110bbbbb bbbbbbbb bbbbbbbb
NOTE b……b is the binary value of ECI assignment number.		

EXAMPLE

Assume data to be encoded are Greek letters "ΑΒΓΔΕ", using character set ISO/IEC 8859-7 (ECI000009).

Data to be encoded: \000009ΑΒΓΔΕ (character values are $A1_{HEX}$ $A2_{HEX}$ $A3_{HEX}$ $A4_{HEX}$ $A5_{HEX}$)

Bit sequence in symbol:

ECI mode indicator:	1000
ECI Assignment number (000009):	00001001
Mode indicator (byte):	0011
Character count indicator:	0000000000101
Data:	10100001 10100010 10100011 10100100 10100101
Final information bit string:	1000 00001001 0011 0000000000101 10100001 10100010 10100011 10100100 10100101

5.4.11.3 Multi-ECI

Refer to the AIM ECI specification for the rules defining the effect of subsequent ECI designator in an ECI data segment. (ECI assignment numbers are 000000 to 000898.)There can be multiple ECI modes to encode data. For example, data to which a character set ECI has been applied may be subject to encryption or compaction using a transformation ECI which will co-exist with the initial ECI (ECI assignment number more than 000898), or a second character set ECI will have the effect of terminating the first ECI and starting a new ECI segment. Where any ECI designator appears in the data, it shall be encoded in the Han Xin Code symbol in accordance with 5.4.11.2 and shall commence a new mode segment.

5.4.12 Unicode mode

5.4.12.1 General

Unicode mode represents the UTF 8 message in Han Xin Code.

For the input data, it is composed of characters which could be represented by 1 byte, 2 bytes, 3 bytes or 4 bytes. In the Unicode mode, the input data is analysed by using the following self-adaptive data analysis algorithm. Firstly, divide and combine the input data into the 1, 2, 3 , or 4 byte pattern pre-encoding sub-sequences, and secondly apply a run-length data compression algorithm to encode eath sub-sequences of the input data . The self-adaptive analysis and run-length encoding process is described in 5.4.12.3. The source code of Unicode mode encoding shall be as listed in the Annex N.

5.4.12.2 Mode indicator and terminator

The mode indicator of Unicode mode is $(1001)_{bin}$ and the mode terminator of Unicode mode is $(1111)_{bin}$.

5.4.12.3 Data analysis and encoding algorithm

a) The Mode Indicator of Han Xin Code Unicode mode is $(1001)_{bin}$.

b) Use following steps to analysis encoded data:

1) Initially analysis the input data in accordance with the encoding algorithm described in c), and get the initial byte pattern analysis result:

I. Read the first 12 bytes of data to analysis:

i) If the whole data length is less than 12 bytes, go to iii); otherwise, go to next step.

ii) Use 1 byte pattern, 2 bytes pattern, 3 bytes pattern and 4 byte pattern to encode the first 12 bytes data, choose the byte pattern which has the lowest pre-encoding bit rate for single byte (encoding bits / bytes counter) as the initial byte pattern of first 12 bytes. If this byte pattern is 4 bytes pattern, then go to viii); otherwise, go to next step.

iii) Read the first 9 bytes of data, if the whole data length is less than 9 bytes, go to v); otherwise, go to next step.

iv) Use 1 byte pattern, 2 bytes pattern (only analysis 8 bytes) and 3 bytes pattern to pre-encode the first 9 bytes data, choose the byte pattern which has the lowest encoding bit rate for single byte as the initial byte pattern of first 9 bytes. If this byte pattern is 3 bytes pattern, then go to viii); otherwise, go to next step.

v) Read the first 6 bytes of data, if the whole data length is less than 6 bytes, go to vii); otherwise, go to next step.

vi) Use 1 byte pattern and 2 bytes pattern to pre-encode the first 6 bytes data, choose the byte pattern which has the lowest encoding bit rate for single byte as the initial byte pattern of first 6 bytes. If this byte pattern is 2 bytes pattern, then go to viii); otherwise, go to next step.

vii) Use 1 byte pattern as the initial byte pattern of first byte.

viii) Utilise the initial byte pattern in the initial data byte series, set the next analysis position of data to the next byte of the initial byte series.

II. For next analysis position of data, if reach the end of data, the initially analysis is over; otherwise, read the next 12 bytes of data, to analysis:

i) If the whole data length from next analysis position is less than 12 bytes, go to iii); otherwise, go to next step.

ii) Use 1 byte pattern, 2 bytes pattern, 3 bytes pattern and 4 byte pattern to pre-encode the next 12 bytes data, choose the byte pattern which has the lowest encoding bit rate for single byte (encoding bits / bytes counter) as the initial byte pattern of next 12 bytes. If this byte pattern is 4 bytes pattern, then go to viii); otherwise, go to next step.

iii) Read the next 9 bytes of data, if the whole data length from next analysis position is less than 9 bytes, go to v); otherwise, go to next step.

iv) Use 1 byte pattern, 2 bytes pattern (only analysis 8 bytes) and 3 bytes pattern to encode the next 9 bytes data, choose the byte pattern which has the lowest encoding bit rate for single byte as the initial byte pattern of the 9 bytes. If this byte pattern is 3 bytes pattern, then go to viii); otherwise, go to next step.

v) Read the next 6 bytes of data, if the whole data length from next analysis position is less than 6 bytes, go to vii); otherwise, go to next step.

vi) Use 1 byte pattern and 2 bytes pattern to encode the next 6 bytes data, choose the byte pattern which has the lowest encoding bit rate for single byte as the initial byte pattern of next 6 bytes. If this byte pattern is 2 bytes pattern, then go to viii); otherwise, go to next step.

vii) Use 1 byte pattern as the initial byte pattern of the next byte.

viii) If the initial byte pattern of the data series is different than the previous initial byte pattern of the previous data byte series, set the initial byte pattern as data patern of the data series, move the next analysis position of data by adding the length of the byte series, go back to II; otherwise, if the initial byte pattern of the data byte series is same as the previous initial byte pattern of the previous data, analyze and decide whether to combine the two data byte series into one or not. The data combination analysis is to calculate and compare the encoding bits for using two separate byte patterns and the encoding bits for integrating to one single byte series:

— If the encoding bits for integrating to one single byte series is not greater than the encoding bits using two separate byte patterns, then do not change the previous byte

pattern to encode, but notice to change the length, difference and minimum values of each byte for the byte pattern;

— If the encoding bits for integrating to one single byte pattern is greater than the encoding bits for using two separate byte patterns, then start a new byte pattern to encode;

— For both of the above two situation, move the next analysis position by adding the length of the new analyzed byte pattern.

2) Optimize the initial byte pattern analysis result by following steps:

I. If the data sequence and the previous data sequence use the same byte pattern:

Calculate the encoding bits of the two adjacent dada sequence with two separate byte patterns (different length, difference and minimum values)and the encoding bits of the two data sequence with single byte pattern:

— If the encoding bits for integrating to one single byte pattern is greater than the encoding bits for using two separate byte patterns, then do not need to change the byte pattern;

— If the encoding bits for integrating to one single byte pattern is not greater than the encoding bits for using two separate byte patterns, then integrate the two separate byte patterns to one single byte pattern, but notice to change the length, difference and minimum values of each byte for the byte pattern.

II. If total byte length of two byte patterns is less than 16:

Calculate the encoding bits of the two adjacent dada sequence with two separate byte patterns (different length, difference and minimum values)and the encoding bits of the two data sequence with one local single byte pattern(1 byte mode):

— If the encoding bits for integrating to one local single byte pattern is not less than the encoding bits for using two separate byte patterns, then do not need to change the byte pattern;

— If the encoding bits for integrating to one local single byte pattern is less than the encoding bits for using two separate byte patterns, then integrate the two separate byte patterns to one local single byte pattern

III. Repeat above steps described in this step until there is no more optimization can be made.

c) Utilise the analysis result by Step 2, use following steps to encode the input data:

1) For each data sequence of 1, 2, 3, or 4 byte pattern, choose the relevant byte pattern indicator as the start bits of the encoding (see Table 6).

Table 6 — Byte pattern indicators in Unicode mode

Byte pattern	Byte pattern indicator
1 byte pattern	$(0001)_{bin}$
2 bytes pattern	$(0010)_{bin}$
3 bytes pattern	$(0011)_{bin}$
4 bytes pattern	$(0100)_{bin}$
The terminator of Unicode mode	$(1111)_{bin}$

2) Utilise the encoding format in Table 7 to encode the byte pattern counter of data sequence of this byte pattern (the byte pattern counter is based on byte pattern, not the byte length, for example 3 bytes pattern with counter 2 has 6 bytes):

Table 7 — Encoding format of the byte pattern counter in Unicode mode

Counter range	Encoding bits	Encoding format
0 to 7	4	0XXX
8 to 63	8	10XX XXXX
64 to 511	12	110X XXXX XXXX
512 to 4095	16	1110 XXXX XXXX XXXX
4096 to 32767	20	1111 0XXX XXXX XXXX XXXX

3) Find the minimum of each 1, 2, 3, or 4 bytes groups, calculate and recode all the differences of each byte positions of the 1, 2, 3, or 4 byte groups compared to the minimum. Add 1,2, 3, or 4 4-bit difference length identifier to the encoding bit stream to encode how many bits will be enough to encode the differences of each byte position compared to the minimum values for each byte positions of each 1, 2, 3 or 4 bytes patterns.

   For example:

   4 bytes pattern, the each byte group needs 2 bits, 3 bits, 6 bits and 8 bits to encode the difference in order, the encoding here is:

   0010 0011 0110 1000

4) Add 1, 2, 3 or 4 8-bit encoding to encode the minimum values for each byte groups. if all data of one byte position is all same (0 bit differences), then this minimum value for this byte position is the value for all data of this byte position.

5) Add differences encoding bits calculated by c)3) to encode difference value of each 1, 2, 3, or 4 byte group of for each byte position from begining of this byte pattern to the end. If all data of one byte position is all same (0 bit differences), then do not need to encode the difference of this byte position.

d) Add the mode terminator of Unicode mode of Han Xin Code - $(1111)_{bin}$, at the end of the encoding of input data.

5.4.13 GS1 mode encoding

5.4.13.1 General

GS1 mode encode GS1 data of GS1 system represented by following structure:

AI_1Data_1 ..... AI_iData_i<FNC1> ..... AI_nData_n

AI means Application Identifier which shall conform to the GS1 General Specification.

Data means the Data of specific Application Identifier AI defined in GS1 General Specification.

5.4.13.2 Data representation rule

When representing the above GS1 data in Han Xin Code, split the GS1 data into several GS1 data segments:

AI_1Data_1	.....	AI_iData_i<FNC1>	.....	AI_nData_n
GS1 data segment 1	.....	GS1 data segment i	.....	GS1 data segment n

where

$$Segment_i = AI_iData_i\ ,\ i = 1, 2, \ldots, m$$

And each GS1 data segment has a AI_i key , and Datai is the value associated with the key AI_i. with the $Data_i$ value of this AI_i key. According to the GS1 specificatons, all predefined length GS1 data strings should be preceded, followed by all non-predefined length GS1 data strings. If more than two non-predefined length strings are used at the end of the whole string, then <FNC1> should be added in between. The separator character <FNC1> is placed immediately after a non-predefined length data segment and is followed by the GS1 Application Identifier of the next data segment. However, If a non-predefined length data segment is the last data segment to be encoded, the <FNC1> should not be used.

5.4.13.3 Mode indicator and terminator

Mode indicator of GS1 mode is $(1110\ 0001)_{bin}$, and mode terminator of GS1 mode is $(1111\ 1111)_{bin}$.

5.4.13.4 Encoding algorithm

a) Use $(1110\ 0001)_{bin}$ to indicate the start of GS1 mode, goto b), etc.

b) Combine the GS1 data segments into data segments groups separated by <FNC1>.

c) Apply the standard Han Xin Code encoding algorithm to the first segments group or the whole message if there is no need to add FNC1.

d) If there are two or more groups, select the first two adjacent GS1 data segment groups separate by <FNC1>, If the leading encoding scheme before the <FNC1> are numeric mode, treat the <FNC1> as a numeric extension which is encode as “1111101000”,then add the <FNC1> encoding into the encoding bits by padding “1111101000” into the encoding bit string, and continue Numeric encoding till the end of the following group or the first character other than 0~9. If the leading encoding scheme is Text mode, first end the Text mode by using Text mode terminator, then add the numeric mode indicator “0001” to start the numeric mode before the <FNC1> encoding, pad the “1111101000” encoding, and utilise the Han Xin Code encoding algorithm till the end of the following group. Then go to step 5

e) Continue the Step 3 till all the groups are encoded, and add the GS1 mode terminator.

EXAMPLE 1

For the GS1 data (01)03453120000011 (17)191125 (10)ABCD1234

a) Use (**1110 0001**)bin to indicate the start of GS1 mode;

b) combine the data segments in to groups. There is no need to add the <FNC1>between the data element strings, so there is only one group:

0103453120000011171912510ABCD1234

c) encode the group with the standard Han Xin Code algorithm.

For 0103453120000011171912510ABCD1234, it could be divided into two string, and encoded with Numeric and Text mode separately.

I. For “010345312000001117191125 10”, the encode process is:

i) Separate the string into groups of three digits: *010 345 312 000 001 117 191 125 10*

ii) Start with change to numeric mode (**0001**) and choose the Terminator: 1111111110

iii) Convert each group into its binary equivalent:

010→0000001010

345→0101011001

312→0100111000

000→0000000000

001→0000000001

117→0001110101

191→0010111111

125→0001111101

10→0000001010, then append the binary terminator (1111111110)

iv) Then the encoding binary sequence for "0103453120000011719112510" is **0001** 0000001010 0101011001 0100111000 0000000000 0000000001 0001110101 0010111111 0001111101 0000001010 1111111110

II. For "ABCD1234", change to text mode (0010), then:

i) Encode the characters in the Text1 sub-mode:

A→001010

B→001011

C→001100

D→001101

1→000001

2→000010

3→000011

4→000100

Then append text mode terminator (111111)

ii) The encoding binary sequence for "ABCD1234" is **0010** 001010 001011 001100 001101 000001 000010 000011 000100 111111.

iii) The final encoding binary sequence is

1110 0001 0001 0000001010 0101011001 0100111000 0000000000 0000000001 0001110101 0010111111 0001111101 0000001010 1111111110 **0010** 001010 001011 001100 001101 000001 000010 000011 000100 111111 11111111.

EXAMPLE 2

For the GS1 data(01)03453120000011 (17)191125 (10)ABCD1234 (21)10,

a) Use (**1110 0001**)bin to indicate the start of GS1 mode;

b) Combine the data segments into groups. <FNC1> should be added between the AI (10) and AI(21) data element strings, so there are two groups:

0103453120000011719112510ABCD1234<FNC1>2110

c) Encode the first group with the standard Han Xin Code algorithm (same as last example).

For "0103453120000011719112510ABCD1234", it could be encoded as "**0001** 0000001010 0101011001 0100111000 0000000000 0000000001 0001110101 0010111111 0001111101 0000001010 1111111110 **0010** 001010 001011 001100 001101 000001 000010 000011 000100 111111" .

d) For <FNC1>2110, as for the leading encoding scheme is Text mode and the characters following <FNC1> are 0~9, it should be encoded as:

I. Choose the Numeric mode indicator "**0001**", and the Terminator "1111111101"

II. <FNC1>should be encoded as "1111101000"

III. 2110 should continue the Numeric encoding, and the binary encoding is “0011010011 0000000000”.

IV. The final binary encoding sequence is

1110 0001 0001 0000001010 0101011001 0100111000 0000000000 0000000001 0001110101 0010111111 0001111101 0000001010 1111111110 **0010** 001010 001011 001100 001101 000001 000010 000011 000100 111111 **0001** 1111101000 0011010011 0000000000 1111111101 11111111

5.4.14 URI mode

5.4.14.1 General

Nowadays, Uniform Resource Identifier (URI) is widely used in bussiness and daily life of every person. URI mode indicate the data represented in Han Xin Code is Uniform Resource Identifier (URI) reference to RFC 3986.

URI Mode is a highly efficientient encoding scheme to encode URI string reference to RFC 3986. By recognizing the characteristics of RFC 3986, URI Mode use 3 charsets (URI-A, URI-B and URI-C),which should conform to Annex M, and a Percent-Encoding charset to represent URI. In this scheme, URI-A charset includes the most common used characters or URI fragments (such as www., .com, http://, etc.) of URI; URI-B includes other characters defined in RFC 3986 and less common keywords; URI-C includes all characters and URI fragments defined in URI-A and URI-B. The analysis and encoding process is described in 5.4.14.3.

5.4.14.2 Mode indicator and terminator

The mode indicator of URI Mode is $(1110\ 0010)_{bin}$ and the mode terminator of URI Mode is $(111)_{bin}$.

5.4.14.3 Data analysis and encoding algorithm

a) The mode indicator of URI mode of Han Xin Code is $(1110\ 0010)_{bin}$

b) Using following methods and steps to analysis the input URI string:

I. Initially analysis the input URI string by following the rules described as follow in accordance with the charsets defined in c), find and record the initial encodings for each characters or character sequence of the input URI string:

i) If there are two character behind the character “%”, and these “%XX” character sequence meet the requirements of Percent-Encoding (RFC 3986 Section 2.1 Page 12) defined in RFC 3986, utilise Percent-Encoding charset to these “%XX” character sequence。

ii) If the character or character sequence can be encoded by using URI-A charset and URI-C charset, utilise URI-A charset as first choice.

iii) If the character or character sequence can be encoded by using URI-B charset and URI-C charset, utilise URI-B charset as first choice.

iv) If there are two ways by used URI-A charset to encode a character or a charset sequence at the same position(s), then utilise the way has the bigger encoding value as first choice.

v) If there are two ways by used URI-C charset to encode a character or a charset sequence at the same position(s), then utilise the way has the bigger encoding value as first choice.

II. After initially analysis described in I, optimize the data analysis result by using the following steps:

i) If the initial encoding for a character or character sequence utilise URI-A charset and URI-B charset is chosen to encode the next character or character sequence, then calculate how many encoding bits will be used to encode these two adjacent character sequence in URI-A and URI-B separately and how many encoding bits will be used by utilising URI-C

charset for the character sequence composed by these two next character sequence. Only if the “combined sequence” encoding bits number be less than or equal to the “separate sequences”, then utilise URI-C charset to encode the character sequence composed by these two adjacent character sequence.

ii) If the initial encoding for a character or character sequence utilise URI-B charset to encode the character sequence and the encoding for the next character or character sequence is URI-A charset, then calculate how many encoding bits will be used to encode these two adjacent character sequence separately and how many encoding bits will be used by utilising URI-C charset for the character sequence composed by these two next character sequence. Only if the “combined sequence” encoding bits number less than or equal to the “separate sequences”, then utilise URI-C charset to encode the character sequence composed by these two adjacent character sequence.

iii) If the initial encoding for a character or character sequence utilise URI-A charset to encode the character sequence and utilise URI-C charset to the next character sequence, then calculate how many encoding bits will be used to encode these two adjacent character sequence separatly and how many encoding bits will be used by utilising URI-C charset for the character sequence composed by these two next character sequence. Only if the “combined sequence” encoding bits number less than will be less than or equal to the “separate sequences”, then utilise URI-C charset to encode the character sequence composed by these two next character sequence.

iv) If the initial encoding for a character or character sequence utilise URI-B charset to encode the character sequence and utilise URI-C charset to the next character sequence, then calculate how many encoding bits will be used to encode these two adjacent character sequences separately and how many encoding bits will be used by utilising URI-C charset for the character sequence composed by these two next character sequence. Only if the “combined sequence” encoding bits number less than or equal to the “separate sequences”, then utilise URI-C charset to encode the character sequence composed by these two next character sequence.

v) If the initial encoding for a character or character sequence utilise URI-C charset to encode the character sequence and utilise URI-A charset to the next character sequence, then calculate how many encoding bits will be used to encode these two next character sequence separately and how many encoding bits will be used by utilising URI-C charset for the character sequence composed by these two next character sequence. Only if the “combined sequence” encoding bits number less than or equal to the “separate sequences”, then utilise URI-C charset to encode the character sequence composed by these two next character sequence.

vi) If the initial encoding for a character or character sequence utilise URI-C charset to encode the character sequence and utilise URI-B charset to the next character sequence, then calculate how many encoding bits will be used to encode these two next character sequences and how many encoding bits will be used by utilising URI-C charset for the character sequence composed by these two next character sequence. Only if the “combined sequence” encoding bits number less than or equal to the “separate sequences”, then utilise URI-C charset to encode the character sequence composed by these two next character sequence.

vii) Repeat above steps described in this step until there is no more optimization can be made.

c) Utilise the analysis result by b), use the indicators in Table 8 and the corresponding charsets to encode the input URI string:

Table 8 — Encoding indicators of each charsets in URI mode

Charset	Charset indicator
URI-A	$(001)_{bin}$
URI-B	$(010)_{bin}$
URI-C	$(011)_{bin}$
Percent-Encoding	$(100)_{bin}$
URI Mode Teminator	$(111)_{bin}$
NOTE 1 URI-A charset is defined in Table M.1. NOTE 2 URI-B charset is defined in Table M.2. NOTE 3 URI-C charset is defined in Table M.3. NOTE 4 Percent-Encoding: For the two characters "XX" followed to the character "%", and the "XX" meet the requirements of hex "00" to "FF" or "00" to "ff", use Percent-Encoding charset to encode the "%XX" character sequence. In the encoding process of Percent-Encoding, after the charset indicator $(100)_{bin}$ add a 8-bit counter to encode the length of the number of how many Percent-Encoding sequences are, after the counter use 8-bit binary string to encode each XX.	

d) The mode terminator of URI mode of Han Xin Code is $(111)_{bin}$.

The C implementation source codes for encoding and decoding are listed in Annex O.

EXAMPLE 1

For the URI http://www.example.com

a) The URI mode indicator is $(\mathbf{1110\ 0010})_{bin}$

b) Analysis of the input URI string:

http://www.example.com

I. Initially analyse the input URI strings:

According to the algorithm, the input URI string could be encoded by URI-A charset(see Table 9), which is started with the URI- A mode indicator (**001**).

Table 9 — URI mode "http://www.example.com" encoding

Character/URI fragment	Encoding Value	Encoding (bits)
http://	49	110001
www.	56	111000
e	4	000100
x	23	010111
a	0	000000
m	12	001100
p	15	001111
l	11	001011
e	4	000100
.com	57	111001
Terminator of URI-A	63	111111

II. Since only URI-A charset is used to encode the whole URI, there is no need to optimize it with other URI character sets.

c) Add the URI mode Terminator (111), then the final encoding binary sequence is (**1110 0010 001** 110001 11100 0 000100 010111 000000 001100 001111 001011 000100 111001 111111 111)$_{bin}$

EXAMPLE 2

For the URI: dictionary.cambridge.org/zhs/词典/英语/soil?q=soil

The URL encoding is: "https://dictionary.cambridge.org/zhs/%E8%AF%8D%E5%85%B8/%E8%8B%B1%E8%AF%AD/soil?q=soil"

The steps for encoding follow.

a) The mode indicator is (1110 0010)$_{bin}$

b) Analysis of the input URI string:

https://dictionary.cambridge.org/zhs/%E8%AF%8D%E5%85%B8/%E8%8B%B1%E8%AF%AD%E9%98%BF%E6%8B%89%E4%BC%AF%E8%AF%AD/soil?q=soil

I. Initial analysis of the input URI strings:

According to the algorithm, the input URI string could be encoded by using a combination of the URI-A charset and the Percent-Encoding mode.

II. The string "https://dictionary.cambridge.org/zhs/" could be encoded in accordance with Table 10.

Table 10 — URI mode "https://dictionary.cambridge.org/zhs/" encoding

Character/URI fragment	Encoding value	Encoding (bits)
http://	49	110001
d	3	000011
i	8	001000
c	2	000010
t	19	010011
i	8	001000
o	14	001110
n	13	001101
a	0	000000
r	17	010001
y	24	011000
.	36	100100
c	2	000010
a	0	000000
m	12	001100
b	1	000001
r	17	010001
i	8	001000
d	3	000011
g	6	000110

Table 10 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
e	4	000100
.org	60	111100
/	37	100101
z	25	011001
h	7	000111
s	18	010010
/	37	100101
Terminator of URI-A	63	111111

III. The Percent-Encodings

The ”%” sequence “%E8%AF%8D%E5%85%B8” is encoded by the Percent-Encoding charset (**100**). There are 6 “%HH” formatted bytes, so set the counter to be “06”$_{HEX}$, and the encoding binary sequence for “%E8%AF%8D%E5%85%B8” is

“**100** 00000110 11101000 10101111 10001101 11100101 10000101 10111000”

The “/” is encoded in URI-A charset, and the encoding binary sequence for “/” is

“**001** 100101 111111”;

The ”%” sequence “%E8%8B%B1%E8%AF%AD” is encoded by returning to Percent-Encoding charset (**100**). There are 6 “%HH” formatted bytes, so set the counter to be “06”$_{HEX}$, and the encoding binary sequence for “%E8%8B%B1%E8%AF%AD” is

“**100** 00000110 11101000 10001011 10110001 11101000 10101111 10101101”

IV. The rest of the URI “/soil?q=soil ”could also be encoded by URI-A charset.(see Table 11)

Table 11 — URI mode “/soil?q=soil” encoding

Character/URI fragment	Encoding value	Encoding (bits)
/	37	100101
s	18	010010
o	14	001110
i	8	001000
l	11	001011
?	43	101011
q	16	010000
=	45	101101
s	18	010010
o	14	001110
i	8	001000
l	11	001011
Terminator of URI-A	63	111111

V. As for only URI-A charset and Percent charset is chosen to encode the whole URI, there is no need to optimize it.

c) Add all the indicator before the encoding binary sequence.

d) Add the URI mode Terminator , and the final encoding binary sequence is (**1110 0010 001** 000011 001000 000010 010011 001000 001110 001101 000000 010001 011000 100100 000010 000000 001100 000001 010001 001000 000011 000110 000100 111100 100101 011001 000111 010010 100101 111111 **100** 00000110 11101000 10101111 10001101 11100101 10000101 10111000 **001** 100101 111111 **100** 00000110 11101000 10001011 10110001 11101000 10101111 10101101 **001** 100101 010010 001110 001000 001011 101011 010000 101101 010010 001110 001000 001011 111111 **111**)$_{bin}$

5.4.15 Mixed modes encoding

Han Xin Code supports any mixing of data sequence of the Numberic, Text, ECI, Binary Bytes and 4 GB18030 modes. The definition of encoding rules used in mixed modes is as follows.

There is the option for a symbol to contain sequences of data in one mode and then to change modes if the data content requires it. Table illustrates the structure of data containing n segments.

Applied to the original data, hybrid encoding rules are designed to transfer information encoding from one mode to another. The basic hybrid encoding structure is as Table 12.

Table 12 — Format of mixed mode data

Segment 1 (mode except Binary Byte mode	mode 1 indicator	data 1	mode terminator 1
Segment 2 (Binary Byte mode)	mode indicator 2	data count indicator 2	data 2
Segment 3 (mode except Binary Byte mode)	mode indicator 3	data 3	mode terminator 3
...	...		
Segment n	mode indicator n	data n	mode terminator n

NOTE Only binary mode applies count indicator while other encoding modes apply mode terminator to denote the mode's end.

Numbers and common Chinese characters may be encoded in several modes. The methods of suitable encoding mode selection are:

1. For numeric data, if the preceding mode of numeric data to be encoded is in any other mode except Text mode, Numeric mode will be selected. If the preceding mode is Text mode, the numeric data will be encoded in following steps:

   a. If in sub-mode Text1 and its length is more than 11 digits, Text encoding mode will be ended and encoding switched to Numeric mode. Otherwise, Text mode will remain in effect.

   b. If in Text2 sub-mode of Text mode and its length is more than 8 digits, Text encoding mode will be ended and encoding switched to Numeric mode. Otherwise, Text mode will remain in effect.

2. For common Chinese characters in GB18030 2-byte region, its hybrid encoding rules are as follows:

   a. For Common Chinese Characters in Region One, if the preceding encoding mode is any one except GB18030 2-byte Region mode, Common Character In Region One encoding mode will be applied. However, if the previous mode is GB18030 2-byte Region and its length is more than 11 characters, encoding will be switched to Common Chinese Character in Region One encoding mode. Otherwise, GB18030 2-byte encoding mode will remain in effect.

b. For Common Chinese Characters in Region Two, if the preceding encoding mode is any one except GB18030 2-byte Region, Common Character In Region Two encoding mode will be applied. However, if the preceding mode is GB18030 2-byte Region and its length is more than 11 characters, it will shift to Common Chinese Character in Region Two encoding mode. Otherwise, GB18030 2-byte encoding mode will remain in effect.

5.5 Error detection and correction

5.5.1 General

Han Xin Code symbols employ Reed-Solomon error correction.

According to the symbol version and error correction level, divide information codewords sequence into blocks (see Annex B) to generate an error correction codewords sequence per block.

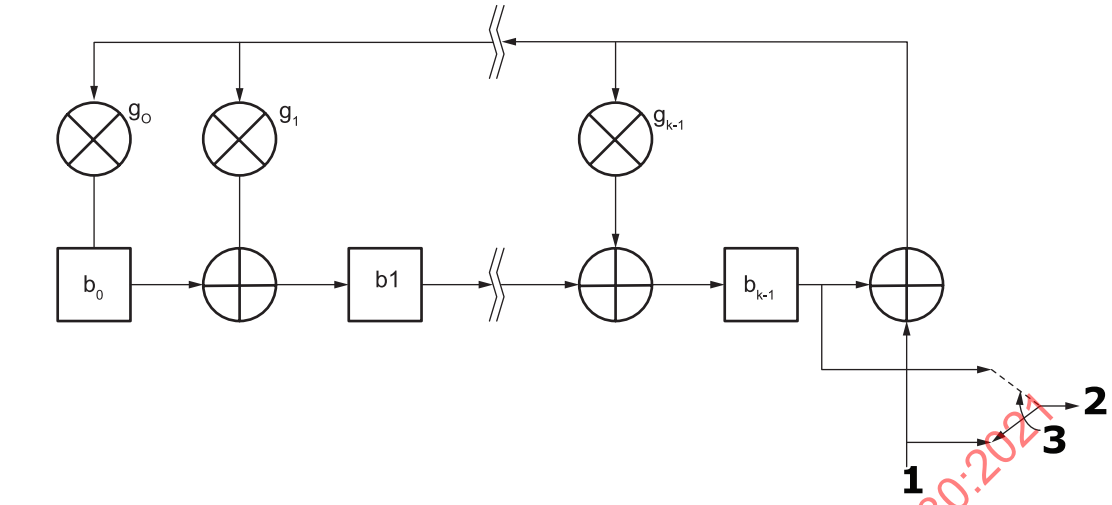
The polynomial arithmetic for Han Xin Code shall be calculated using bit-wise modulo 2 arithmetic and byte-wise modulo $(101100011)_{bin}$ (decimal 355) arithmetic. This is based on the Galois field of 2^8 with $(101100011)_{bin}$ representing the field's prime modulus polynomial:$x^8+x^6+x^5+x+1$. Thirty-two different generator polynomials which shall in accordance with Annex D are used for generating the appropriate error correction codewords.

5.5.2 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 algorithm (see Annex D).

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 in the block.

The error correction codewords can be generated by using the division circuit as shown in Figure 18. The registers b_0 through b_{k-1} are initialized 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 ... n+k clock pulses), with the switch in the up position, the error correction codewords are generated by flushing the registers in order and complementing the output 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.



Key

1	Input
2	Output
3	Switch
⊕	GF(2^8) Addition
⊗	GF(2^8) Multiplication

Figure 18 — Error correction codeword encoding circuit

Divide the information codewords sequence into corresponding blocks according to Table B.1 for Han Xin Code. Error correction codewords are calculated and added behind the information codewords in each block.

The following steps are the generating error correction codewords detailed process of Han Xin Code:

1. Set information codeword polynomial as $m(x)=m_{k-1}\ x^{k-1}+m_{k-2}x^{k-1}+\ldots+m_1x+m_0$. The polynomial coefficient adopts Galois Field GF(2^8), i.e., $m_i \in GF(2^8)$. Polynomial arithmetic uses the bit's modulo 2 Arithmetic and byte's modulo $(101100011)_{bin}$ arithmetic, i.e., setting α as the generator of the GF(2^8), so $\alpha^8+\alpha^6+\alpha^5+\alpha+1=0$.

   The information codewords are the coefficients of the various items of the polynomial. The first information codeword is the coefficient of the highest degree, while the last information codeword is the coefficient of the lowest degree.

2. Calculate the generator polynomial $g(x)$

$$g(x)=\prod_{i=1}^{2t}\left(x-\propto^{i}\right)=x^{2t}+g_{2t-1}x^{2t-1}+\ldots+g_1x+g_0$$

3. The error correction codewords are the remainder of the information codeword polynomial multiplying by x^{2t} and then being divided by $g(x)$, i.e., $r(x)=m(x)x^{2t}\ mod\ g(x)$. The highest degree of the remainder is the first error correction codeword, while the lowest degree coefficient is the last error correction codeword.

4. The data codeword polynomial is $c(x)=m(x)x^{2t}+r(x)$, where the highest degree coefficient of $c(x)$ is the first codeword of data codewords sequence, while the lowest degree coefficient is the last codeword of data codewords sequence.

5.5.3 Error correction capacity

The error correction codewords can correct two types of erroneous codeword, 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 correctable 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 undetected errors is increased. Whenever the number of erasures is more than half the number of error correction codewords, p=3.

According to version and error correction level, information codewords sequence can be divided into one or more blocks, and every block will be processed with error correction algorithm. The total number of codewords, total number of error correction codewords, and the structure and number of the error correction blocks of each level are listed in Annex B.

5.6 User considerations for encoding data in a Han Xin Code symbol

5.6.1 General

Han Xin Code offers flexibility in the way data is encoded. Alternate character sets can be invoked using the ECI protocol.

5.6.2 User selection of error correction level

Han Xin Code symbols offer four levels of error correction. 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 the size of the symbol), and offer different levels of data recovery. The basic features of error correction level are shown in Table .

Table 13 — Error correction levels features

Error correction level	Recovery capacity % (approximation)	Encoding of error correction level
L1	8	$(00)_{bin}$
L2	15	$(01)_{bin}$
L3	23	$(10)_{bin}$
L4	30	$(11)_{bin}$

5.6.3 User selection of mode

Han Xin Code symbols offer eleven modes of encoding (which are specified in 5.4). Generally the modes are used to define the format of the message data.

5.6.4 User selection of Extended Channel Interpretation

The use of an alternate 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 5.4.11) provides the capability to encode data from alphabets other than the ASCII characterssupported by the default interpretation (ECI 000003).

5.6.5 User selection of symbol size

Han Xin Code has eighty-four versions (sizes). The size may be selected to meet the requirement of the application. The sizes are technically specified in 4.2.2.

5.7 Construction of final data bit stream

The following steps shall be followed to construct the data bit stream:

1. Divide the information codewords sequence into n blocks according to the version and error correction level, see Annex B.
2. For each information block, calculate a corresponding block of error correction codewords; see 5.5.2 and Annex D.
3. Assemble the final sequence by taking information and error correction codewords from each block in turn: the information codewords sequence of information block 1; the error correction sequence of error correction block 1; the information codewords sequence of information block 2; the error correction sequence of error correction block 2; the information codewords sequence of information block 3; the error correction sequence of error correction block 3;...; the information codewords sequence of information block n; the error correction sequence of error correction block n.
4. Convert each codeword to 8-bit binary string, thus convert the final data codewords sequence to the final data bit stream.

5.8 Symbol construction

5.8.1 General

Given the codewords sequence obtained in the subclauses 5.2, 5.3, 5.4, 5.5, 5.6 and 5.7 , the Han Xin Code symbol is constructed using the following steps(see Annex K):

1. Place Fixed Pattern in the Han Xin Code symbol;
2. Place data modules in the symbol;
3. Data masking;
4. Place Structural Information in the symbol.

5.8.2 Fixed Pattern placement

A square blank matrix shall be constructed with the number of modules horizontally and vertically corresponding to the version in use. Positions corresponding to the Finder Pattern, Position Detection Pattern separators, Alignment Pattern and Assistant Alignment Patterns shall be filled with either dark modules or light modules as appropriate. Module positions for the Structural Information region shall be left temporarily blank. Annex A defines the positioning of Alignment Pattern.

5.8.3 Data placement

The generated data codewords sequence will first be divided as every 13 codewords as a group. Connect codewords in the same position of each group and form new groups, then link new groups together to generate a new codewords sequence. For example, the final codewords stream of version 9 is $C_1C_2C_3$... C_{136}, where C_i (i=1, 2, 3 ... 136) is represented as codeword, and the final data codewords to be arranged after broken up is $C_1C_{14}C_{27}...C_2C_{15}C_{28}...C_{130}$. The new codewords stream will be encoded into 8-bit binary string and transfer into a sequence to be arrayed following the rule that 1 is dark module and 0 is light module. Place the sequence to be arrayed into the Han Xin Code symbol, row by row from left to right and from up to down. Overleap when confronting Fixed Pattern or Structural Information region and re-break the arrangement when confronting code pattern boundary. Fill in symbol padding bit when there is blank in the pattern.

As illustrated in Figure 19, the bits belonging to the same codeword group (the same color) are dispersed throughout the entire symbol. Horizontal line damage would affect fewer codewords; vertical line damage would affect more codewords.

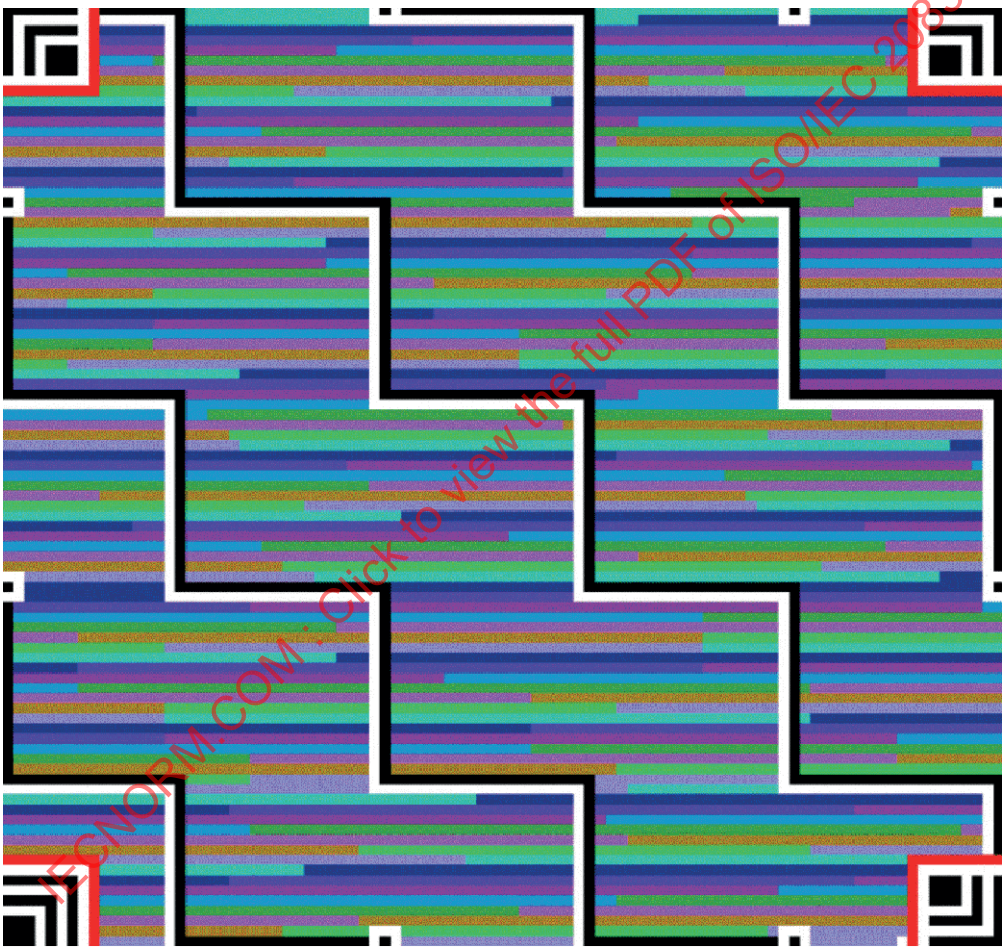


Figure 19 — Arrangement of codeword bits in Han Xin Code symbols

EXAMPLE 1

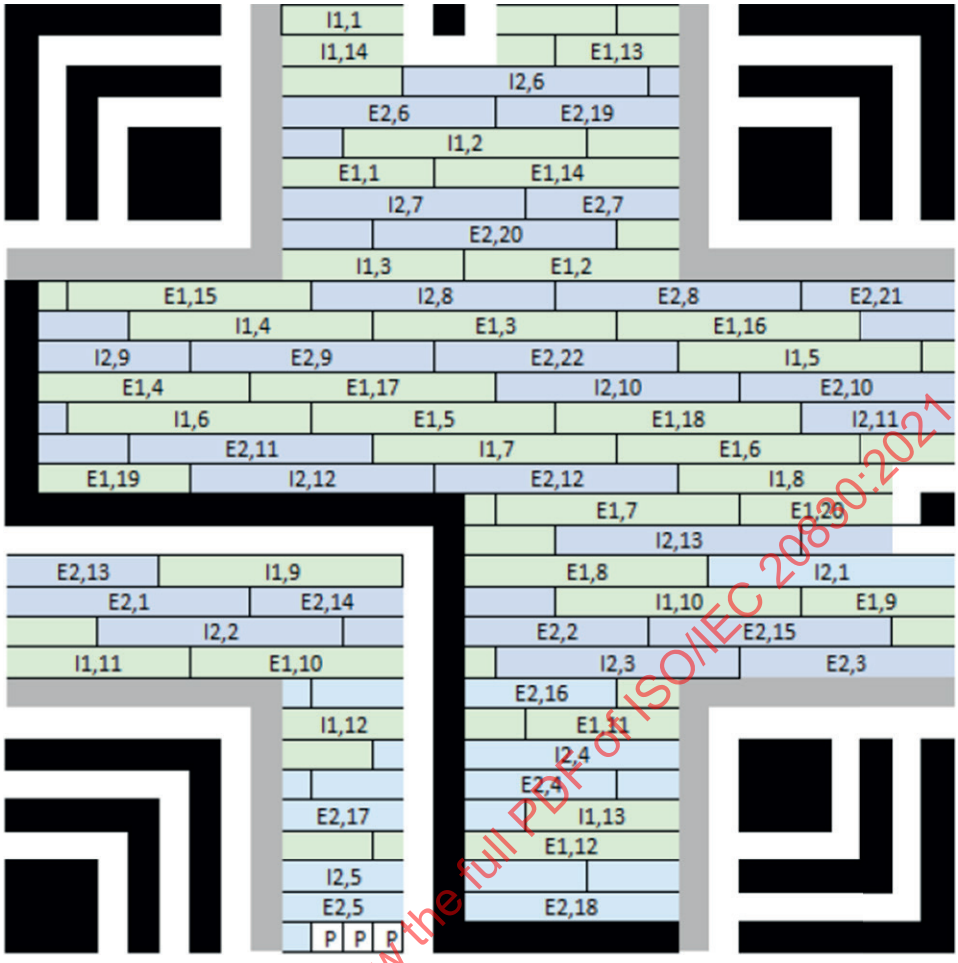


Figure 20 — Arrangement of codeword bits in V5, L4 Han Xin Code symbol

There are two error correction blocks in version 5, L4 Han Xin Code symbol. The first block has 34 codewords which include 14 information codewords (I)and 20 error codewords(E); the second block has 35 codewords which include 13 information codewords (I)and 22 error codewords(E). The codewords series of the two blocks are:

Block 1: “I1,1” “I1,2” … “I1,14” “E1,1” “E1,2” … “E1,20”

Block 2: “I2,1” “I2,2” … “I2,13” “E2,1” “E2,2” … “E2,22”

Linking the two data codeword streams, we got:

“I1,1” “I1,2” … “I1,14” “E1,1” “E1,2” … “E1,20” “I2,1” “I2,2” … “I2,13” “E2,1” “E2,2” … “E2,22”

After the mod 13 randomization, we got the final codeword stream to be placed in the symbol:

“I1,1” ”I1,14” ”E1,13” ”I2,6” ”E2,6” ”E2,19” ”I1,2” ”E1,1” ”E1,14” ”I2,7” ”E2,7” ”E2,20” ”I1,3” ”E1,2” ”E1,15” ”I2,8” ”E2,8” ”E2,21” ”I1,4” ”E1,3” ”E1,16” ”I2,9” ”E2,9” ”E2,22” ”I1,5” ”E1,4” ”E1,17” ”I2,10” ”E2,10” ”I1,6” ”E1,5” ”E1,18” ”I2,11” ”E2,11” ”I1,7” ”E1,6” ”E1,19” ”I2,12” ”E2,12” ”I1,8” ”E1,7” ”E1,20” ”I2,13” ”E2,13” ”I1,9” ”E1,8” ”I2,1” ”E2,1” ”E2,14” ”I1,10” ”E1,9” ”I2,2” ”E2,2” ”E2,15” ”I1,11” ”E1,10” ”I2,3” ”E2,3” ”E2,16” ”I1,12” ”E1,11” ”I2,4” ”E2,4” ”E2,17” ”I1,13” ”E1,12” ”I2,5” ”E2,5” ”E2,18”

The layout of the codeword stream is illustrated in Figure 20 followed by 3 padding bits.

5.8.4 Masking

5.8.4.1 General

For reliable Han Xin Code reading, it is preferable for the proportion of dark and light modules closely to 1:1 in the symbol. The bit pattern 1:1:1:1:3 or 3:1:1:1:1 particularly found in the Finder Pattern should be avoided in other areas of the symbol as much as possible. Four data masking patterns are available. The data mask that provides the best solution should be used.

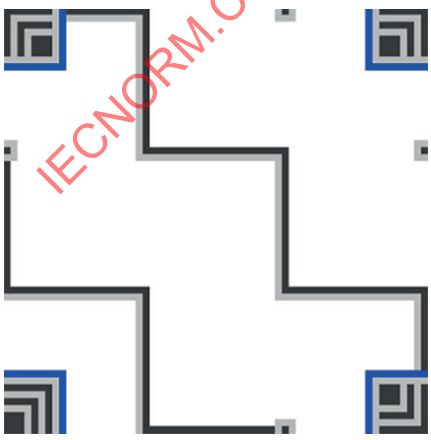
Apply the data masking patterns in turn o the encoding region of the Han Xin Code symbol. Table shows the data mask pattern reference (binary reference for use in the Structural Information region) and the data mask pattern generation condition. Convert the given module pattern in the encoding region (not including the Fixed Pattern and Structural Information region) with modules corresponding masking patterns successively through the XOR operation.

Table 14 — Masking pattern algorithm

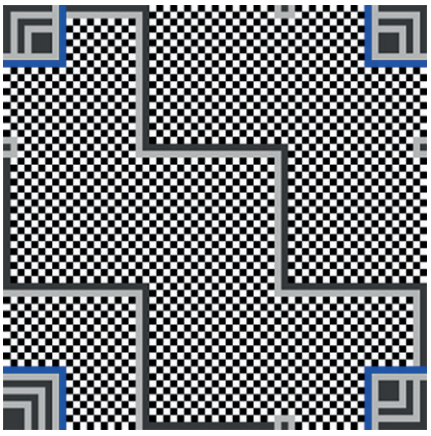
Condition of masking solution	Data mask pattern reference for Han Xin Code
Non-masking	00
$(i+j)$ mod 2=0	01
$((i+j)$mod 3+$(j$ mod 3)) mod 2=0	10
(i mod j +j mod i + i mod 3+ j mod 3) mod 2=0	11
Key i row index of the symbol j column index of the symbol NOTE 1 Both i and j start from 1. (1,1) is the top left corner module of the symbol. NOTE 2 When the masking solution condition is true, the resulting mask bit is 1.	

5.8.4.2 Masking pattern

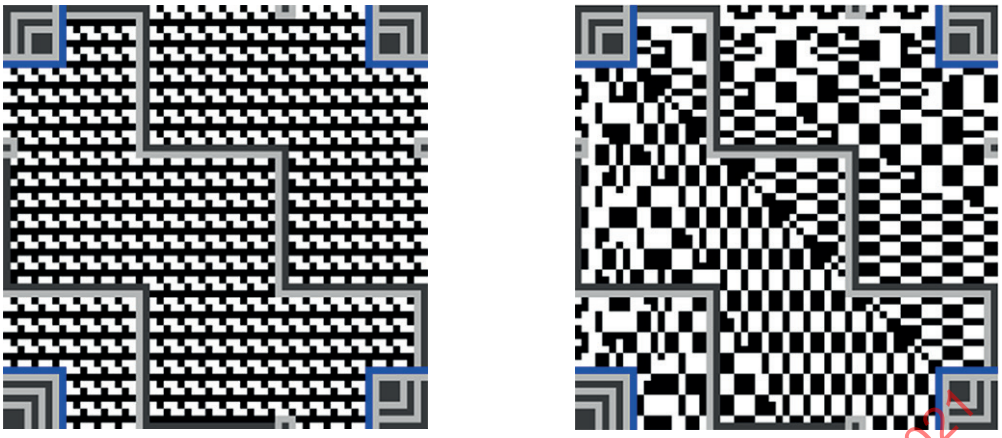
The data masking pattern is generated by defining as dark any module in the encoding region (not including the area reserved for Fixed Pattern and the Structural Information region) for which the condition is true; In Table , i refers to the row position of the module in question and j to its column position in the symbol. Figure 21 shows all data mask patterns of Han Xin Code symbols.



a) 00



b) 01



c) 10 d) 11

Figure 21 — Data masked patterns

5.8.4.3 Evaluation of data masking results

After performing the data masking operation with each data mask pattern in turn for the Han Xin Code symbol, the results shall be evaluated by scoring penalty points for each occurrence of the following features. Table shows the penalty rule of masking solution according to the pattern structure features of the Han Xin Code symbol. In Table , *i* refers to the modules with same color. Select the pattern with the lowest penalty points score.

Table 15 — Penalty rules of data masking results

Occurrence of features	Penalty conditions	Penalty points
1:1:1:1:3 or 3:1:1:1:1 ratio (dark:light:dark: light:dark) pattern of the Position Detection Pattern appeared in row or column, preceded or followed by light area 3 modules wide.		50
Adjacent modules in row or column in same color	number of modules = 3+*i* (*i*≥0)	(3+*i*) × 4

5.8.5 Structural Information placement

The Structural Information of Han Xin Code contains the version number, error correction level and masking solution information, represented by 8-bit, 2-bit and 2-bit binary strings respectively, totalling 12 bits. The error correction of the Structural Information employ Reed-Solomon error correction algorithm in Galois Field GF(2^4) , which shall conform with Annex E, and generates 34-bit Structural Information.

The capacity of Structural Information region of Han Xin Code symbol contains 17×4=68 modules. Method of Structural Information modules placement in the mapping Structural Information region as described in following steps:

1. Place the Structural Information into the upper left corner and the upper right corner of the Structural Information region respectively with "1" represented as dark modules and "0" represented as light modules. Array each Structural Information region by counterclockwise methods as shown in Figure .

2. Place the Structural Information into the lower right corner and the lower left corner of the Structural Information region respectively with "1" represented as dark modules and "0"

represented as light modules. Array each Structural Information region by counterclockwise methods as shown in Figure 22.

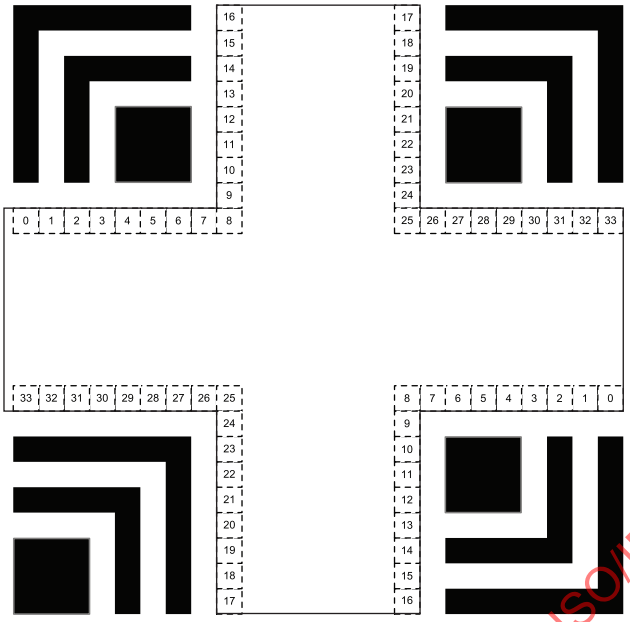


Figure 22 — Structural Information placement

6 Symbol dimensions

6.1 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.
Y dimension:	the height of the module shall be the same as its width.
Position Detection Pattern:	the width of the Position Detection Pattern shall equal 7X.
Alignment Pattern:	the width of the maximum Alignment Pattern shall equal 2X.
Assistant Alignment Pattern:	the width of the Assistant Alignment Pattern shall equal 3X.
	the height of the Assistant Alignment Pattern shall equal 2X.

6.2 Quiet zone

The minimum quiet zone is equal to three module widths on all four sides.

7 User guidelines

7.1 Human readable interpretation

Because Han Xin Code symbols are capable of encoding many characters, a human readable interpretation of the data characters may not be practical. As an alternative, descriptive text rather than the encoded text can accompany the symbol. The character size and font are not specified; the message can be printed anywhere in the area adjacent to the symbol. The human readable interpretation shall not interfere with the symbol itself or the Quiet Zones.

7.2 Autodiscrimination capability

Han Xin Code can be used in an auto discrimination environment with a number of other symbologies. (See Annex F).

7.3 Principle of Han Xin Code symbol printing and scanning

See Annex H for user guiding principle of Han Xin Code symbol printing and scanning.

8 Symbol quality

8.1 General

Han Xin Code symbols shall be assessed for quality using the two-dimensional matrix bar code symbol print quality guidelines defined in ISO/IEC 15415, as augmented and modified below.

8.2 Symbol quality parameters

8.2.1 General

Symbol Quality parameters include decode, symbol contrast, modulation, axial non-uniformity, grid non-uniformity, unused error correction, structural information damage and fixed pattern damage as defined in ISO/IEC 15415 and subclauses 8.2.2, 8.2.3 and 8.2.4 of this document.

8.2.2 Fixed Pattern damage

Fixed Pattern damage shall be measured and graded in accordance with Annex I.

8.2.3 Symbol grade

The symbol grade should be the lowest of the grades for Decode, Symbol Contrast, Modulation, Fixed Pattern Damage, Axial Non uniformity, Grid Non uniformity and Unused Error Correction in an individual image of the symbol.

8.2.4 Grid non-uniformity

The grid that is used to decode the symbol is calculated by using the Finder Pattern, Alignment Pattern and Assistant Alignment Pattern as datum points, as located by the use of the reference decode algorithm (see Clause 10).

The ideal grid is calculated nominally by dividing the distance between the centers of the four squares of the Finder Pattern according to the symbol version to form a grid of nominal module centers. Grid Nonuniformity is calculated as defined in ISO/IEC 15415 based on these two grids.

8.3 Process control measurements

Several tools and methods are available which can perform useful measurements for monitoring and controlling the process of creating Han Xin Code symbols. Such process control measures do not replace formal print quality grading according to 8.2 but may be useful in practice. These include:

1. Symbol contrast readings from a linear bar code verifier.
2. A template overlay to visually check Finder Pattern growth, Alignment Pattern, Assistant Alignment Pattern, and overall symbol size.
3. Determination of axial non-uniformity by physical measurement.

4. Visual inspection for dark module growth and defects.

These tools and methods are described in Annex J.

9 Decoding procedure overview

The decoding steps from reading a Han Xin Code symbol to outputting data characters are the reverse of the encoding procedure. This procedure includes image preprocessing, symbol finding and orientation, Structural Information decoding, sampling grid establishment, release masking, restore data and codewords sequence, error correction decoding, information decoding. Figure 23 shows an outline of the process flow.

1. Image preprocessing: Locate and obtain an image of the symbol. Transfer the image into a series of binary pattern composed of dark and light pixels. Recognize dark and light pixels as an array of "1" (dark) and "0" (light).
2. Symbol finding and orientation: Search the Finder Pattern and determine symbol's position and direction.
3. Structural Information decoding: Extract Structural Information bits in all of the Structural Information regions, construct a grid of module centers, convert to codewords, perform error correction and ascertain the symbol's version, error correction level and masking solution.
4. Sampling grid establishment: Establish a sampling grid and sample Han Xin Code symbol according to version information and other information.
5. Release masking: Release the masking by XOR the encoding region bit pattern with the data masking pattern which has been determined from the Structural Information.
6. Restore the data codeword sequence: Restore the data codeword sequence according to the data placement rules.
7. Error correction decoding: Detect errors using the error correction codewords corresponding to the level information. If any error is detected, correct it.
8. Information decoding: Decode the data codewords sequence and restore original information and output the result.

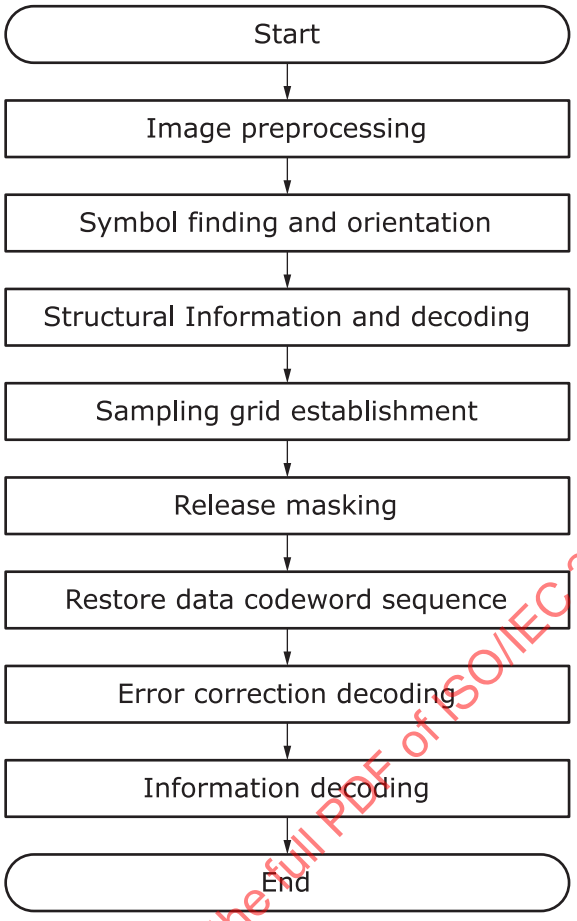


Figure 23 — Decoding process of Han Xin Code

10 Reference decode algorithm for Han Xin Code

10.1 General

This reference decode algorithm finds the symbols in an image and decodes them. This is the decode algorithm used in the determination of symbol quality. This decode algorithm refers to black and white states in the image.

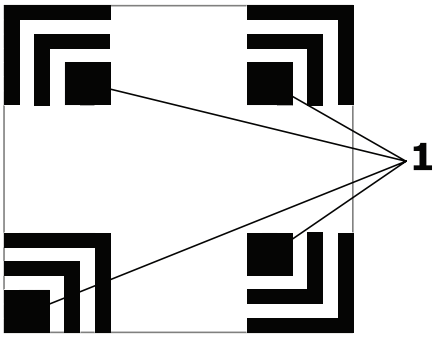
10.2 Image preprocessing

Determine a global threshold according to the method defined in ISO/IEC 15415. Using the global threshold, convert the image to the binary image of a set of dark and light pixels.

10.3 Locate Finder Pattern and determine the orientation

The Finder Pattern in Han Xin Code consists of four Position Detection Patterns located at the four corners of the symbol. The shapes of all Position Detection Patterns are the same; their orientation varies, as shown in Figure 24.

Module widths in each Finder Pattern form a dark-light-dark-light-dark sequence. The ratios of the relative widths of each element are 1:1:1:1:3 or 3:1:1:1:1. For the purposes of this algorithm the tolerance for each of these widths is 0.5 (i.e., a range of 0.5 to 1.5 for the single module block and 2.5 to 3.5 for the three modules square block).

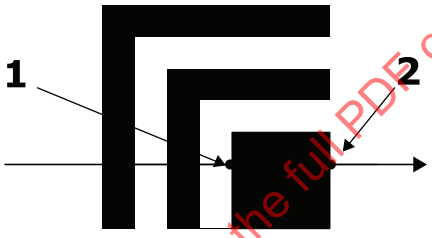


Key

1 Finder Pattern

Figure 24 — Position of Finder Pattern

a) When a candidate area is detected, mark the position of the first and last points A and B respectively at which a line of pixels in the image encounters the outer edges of the Position Detection Pattern (see Figure 25). Repeat this for adjacent pixel lines in the image until all lines crossing the 3×3 dark modules of the Position Detection Pattern in the x axis of the image have been identified.



Key

1 A

2 B

Figure 25 — Scanning line of Position Detection Pattern

b) Repeat step a) for pixel columns crossing the 3×3 dark modules of the Position Detection Pattern in the y axis of the image.

c) Locate the center of the Position Detection Center. Construct a line through the midpoints between the points A and B on the outermost pixel lines crossing the central 3×3 dark module of the Position Detection Pattern in the x axis. Construct a similar line through points A and B on the outermost pixel columns crossing the central 3×3 dark module in the y axis. The central coordinates (x, y) of the Position Detection Pattern is located at the intersection of these two lines.

d) If no central coordinates (x, y) is found, construct new axes x' and y' by rotating the x and y axes 45 degrees counterclockwise, repeat a) and c) in x' axis and y' axis to locate the central coordinates (x', y').

e) Repeat steps a) to d) to locate the central coordinates of the three other Position Detection Patterns.

f) If no Position Detection Center is found, reverse the black and white states of the whole image and repeat steps a) to e) to the reversed image.

g) Determine the rotational orientation of the symbol by analyzing the central coordinates of 1:1:1:1:3 or 3:1:1:1:1 ratio (dark:light:dark:light:dark) pattern of the Position Detection Pattern to identify which pattern is the lower left pattern in the symbol and the angle of rotation of the symbol.

10.4 Structural Information decoding

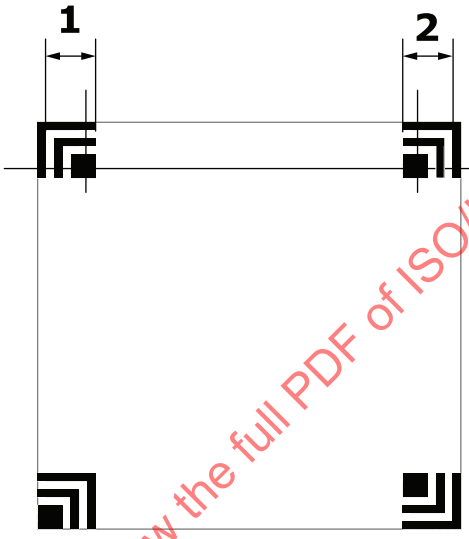
Acquire Structural Information in the Structural Information region and decode them to know the version, error correction level and masking solution of the Han Xin Code symbol.

a) Calculate the nominal Z dimension module width of the symbol:

$X=(W_{UL}+W_{UR})/12$

where W_{UL} and W_{UR} are width of the upper left corner Position Detection Pattern and upper right Position Detection Pattern respectively, as shown in Figure 26.

Perform the same calculation to determine the vertical Z dimension using W_{UR} and W_{LR}.



Key

1 W_{UL}

2 W_{UR}

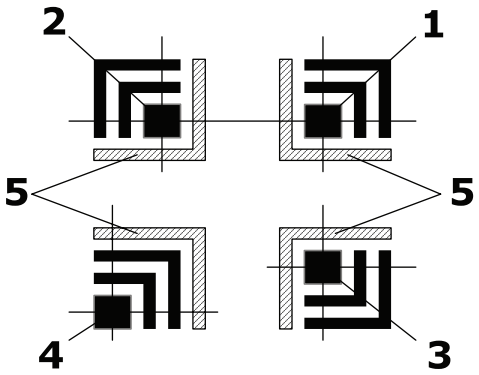
Figure 26 — Upper Position Detection Patterns

b) The Structural Information is decoded as follows:

1) Divide the width W_{UR} of upper right Position Detection Pattern by 6 to calculate the module size CP_{UR}.

$CP_{UR}=W_{UR}/6$

2) Find the guide lines AC and AB from A, B and C, which pass through the centers of the three Position Detection Patterns, as shown in Figure 27. The sampling grid for each module center in the Structural Information region near Position Detection Pattern A and B is determined based on lines parallel to the guide lines, the central coordinates of the Position Detection Patterns, and the module size CP_{UR}, then extract the Structural Information.



Key

1 A
2 B
3 C
4 D
5 Function information

Figure 27 — Position Detection Patterns and Structural Information

3) Determine the version, error correction level and masking solution of Han Xin Code symbol by detecting and correcting errors of the Structural Information based on Annex E. If errors exceeding the error correction capacity are detected, then calculate the pattern width W_{DL} of the lower right and lower left Position Detection Pattern and follow a similar procedure to steps a), b) and c) above to decode the Structural Information extract from the Structural Information region near lower position detection region C and D.

10.5 Establish the sampling grid

Han Xin Code symbols of different versions use different ways to establish a sampling grid.

1. Establishing the sampling grid for Han Xin Code symbols of versions 1 to 3.

   For symbols of versions 1 to 3 without an Alignment Pattern, take the following steps to establish the sampling grid(see Figure 28):

   a) According to the version information obtained in 10.4, determine the horizontal mean distance X between two adjacent module centers and the vertical mean distance Y between two adjacent module centers.

   b) Establish the sampling grid.

      1) Draw lines paralleled with line AB as shown in Figure . The distance between the two adjacent parallel lines is Y. There are 5 lines above line AB, and (n-6) lines below line AB, where n is the number of symbol modules in the symbol.

      2) Draw lines paralleled with line AC. The distance between the two adjacent parallel lines is X. There are 6 lines right to line AC, and (n - 6) lines left to line AC, where n is the number of symbol modules in the symbol.

      3) Find L_x, which is the center-to-center distance of the point C and D, and L_y, which is the center-to-center distance of the point A and C. Divide L_x and L_y by the defined distance of the point C and D to obtain the module pitches CP_x in the lower side and CP_y in the right side (see Figure 28) using the following formula.

         $$CP_x=L_x/AP_{cd}$$

$CP_y = L_y / AP_{ac}$

where AP_{cd} is the distance in modules of the Position Detection Center C and D (see Table A.1).

where AP_{ac} is the distance in modules of the Position Detection Center A and C (see Table A.1).

In the same way, find $L_{X'}$, which is the horizontal distance between the coordinates of A and the coordinates of B, and $L_{Y'}$, which is the vertical distance between the coordinates of B and the coordinates of D. Divide $L_{X'}$ and $L_{Y'}$ by the following formula to obtain the module pitches $CP_{X'}$ in the upper side and $CP_{Y'}$ in the left side.

$CP_{X'} = L_{X'} / AP_{ab}$

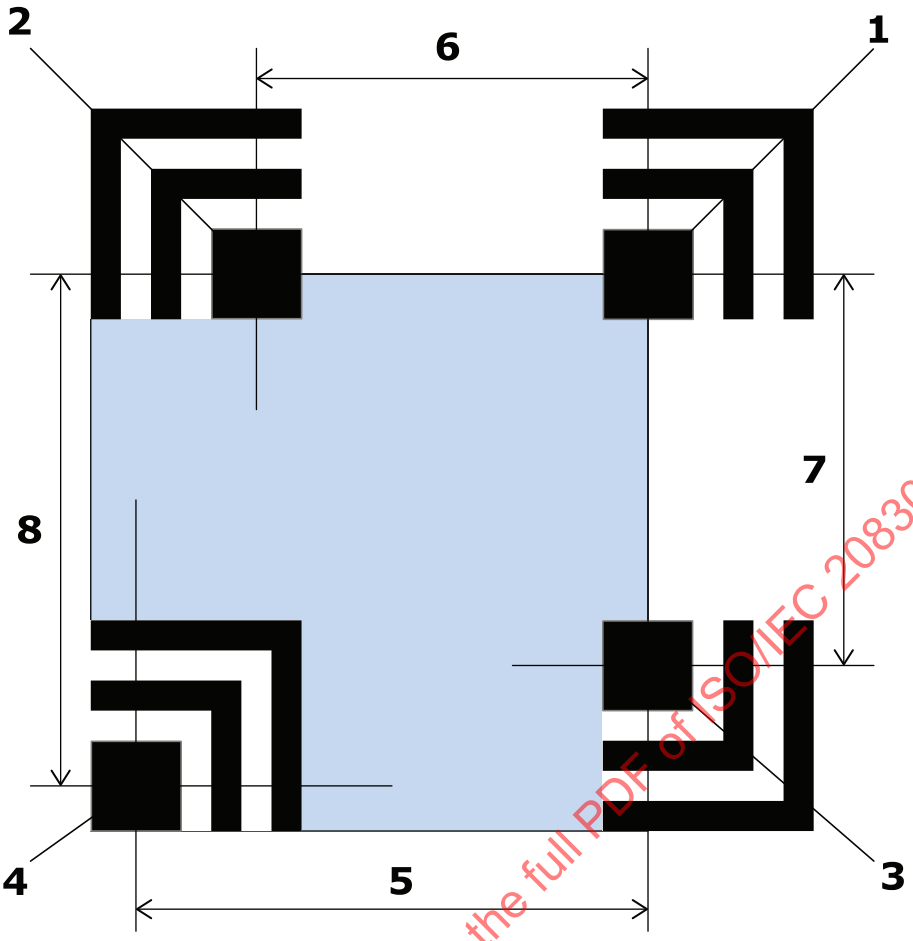
$CP_{Y'} = L_{Y'} / AP_{bd}$

where AP_{ab} is the distance in modules of the Position Detection Center A and B (see Table A.1).

where AP_{bd} is the distance in modules of the Position Detection Center B and D (see Table A.1).

4) Determine the sampling grid covering the shadow area (see Figure 28), and the horizontal lines extending to right cross the lines from step 2. The vertical lines extending to top cross the lines from step 1.

c) Perform sampling as described in 10.6.



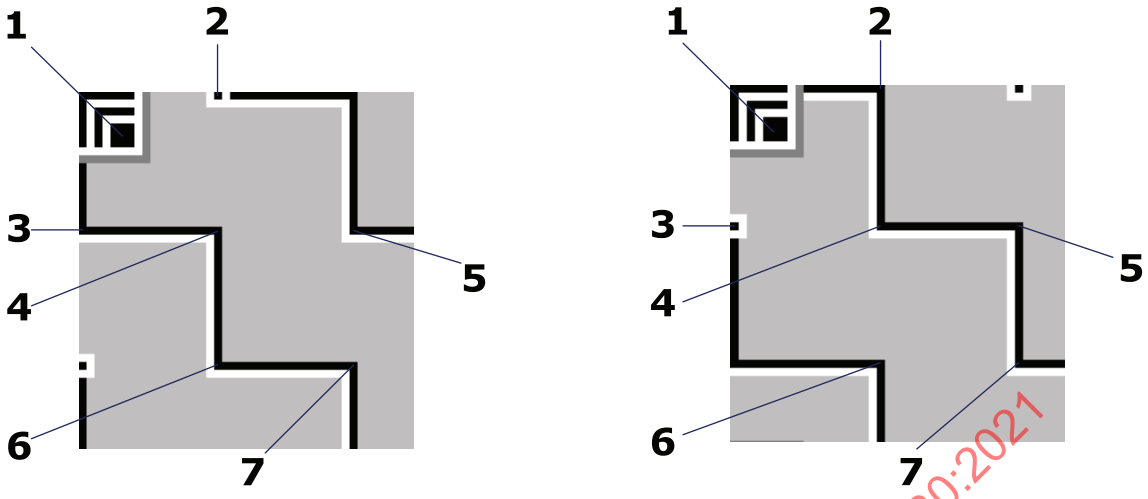
Key

1	A
2	B
3	C
4	D
5	L_x
6	$L_{x'}$
7	L_y
8	$L_{y'}$

Figure 28 — Sampling grid establishment of version 1~3 Han Xin Code

2. Establishing the sampling grid for version 4 and larger symbols in Han Xin Code.

For version 4 or higher, the symbol is divided into several regions by the Alignment Pattern. For Alignment Pattern parameters of different versions see Annex A. The sampling grid for each symbol region should be established separately according to the adjacent Alignment Pattern and Assistant Alignment Patterns. The positions of Alignment Pattern and Assistant Alignment Pattern are different in various versions of Han Xin Code symbols. There are two conditions for the upper left corner region, as shown in Figure 29 a) and b). The following is to introduce the establishing way of sampling grid in the region of upper left corner region by Figure 29 a).



a) The upper left corner region of symbols with Assistant Alignment Pattern located on the top edge

b) The upper left corner region of symbols with Assistant Alignment Pattern located on the left edge

Key

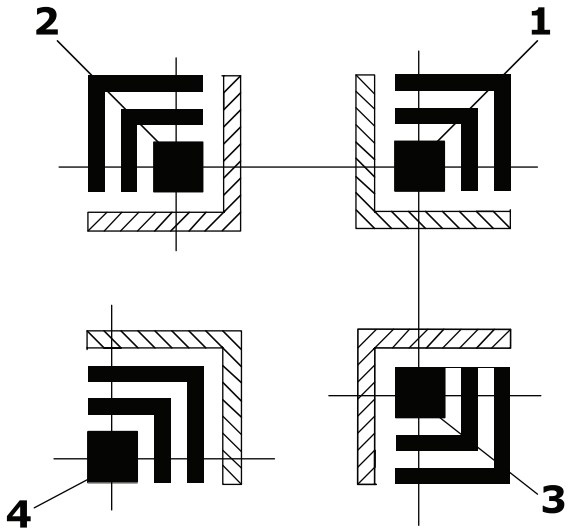
1	P_{UL}
2	P_1
3	P_2
4	P_3
5	P_4
6	P_5
7	P_6

Figure 29 — Position detection pattern with alignment pattern of different versions

a) Divide the pattern width W_{UL} of the upper left Position Detection Pattern by 6 to calculate the module size CP_{UL}.

$$CP_{UL} = W_{UL}/6$$

b) Determine the provisional turning point coordinates P_2 of the Alignment Pattern and central coordinates P_1 of Assistant Alignment Pattern, based on the central coordinates of the upper left corner Position Detection Pattern P_{UL}, the version information, lines parallel to the guide lines $P_{UL}P_{UR}$ and $P_{UR}P_{DR}$ (as shown in Figure 30), and the module size CP_{UL}.



Key

1 P_{UR}
2 P_{UL}
3 P_{DR}
4 P_{DL}

Figure 30 — Four Position Detection Patterns

c) Scan the adjacent 8 white modules starting from the pixel of the provisional center to find the actual central coordinates X_i and Y_i (see Figure 31). Search the alternating light and dark lines of the Alignment Pattern around coordinates P_2 to find the turning point coordinates i.e. the final actual coordinates X_j and Y_j.

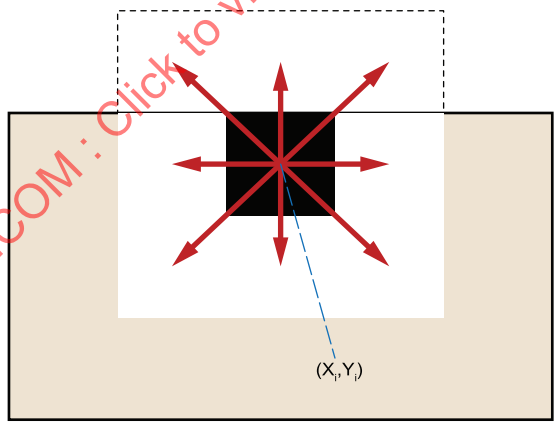


Figure 31 — Center coordinates of Assistant Alignment Pattern

d) Estimate the provisional coordinates P_3 of the Alignment Pattern, based on the central coordinates of the upper left Position Detection Pattern P_{UL} and the actual coordinates of the P_1 and P_2 obtained in c).

e) Find the actual coordinates of P_3 by following the same procedure in c).

f) Determine the module dimension size.

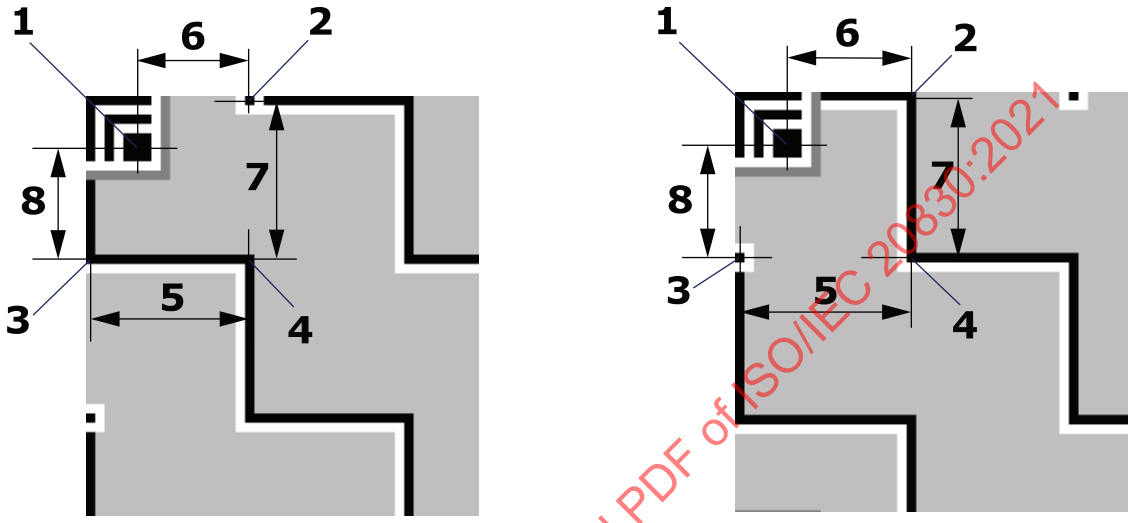
Find L_x, which is the center-to-center distance of the point P_2 and P_3, and L_y, which is the center-to-center distance of the point P_1 and P_3. Divide L_x and L_y by the defined distance of the point P_2 and P_3 to

obtain the module pitches CP_x in the lower side and CP_y in the right side in the upper left corner area of the symbol (see Figure 32) using the following formula:

$CP_x = L_x / AP$

$CP_y = L_y / AP$

where AP is the distance in modules of the turning points of Alignment Pattern adjacent (see Table A.1).



a) The relative positions and distances of the points for regional sampling of the upper left corner region, Assistant Alignment Pattern located on the top edge

b) The relative positions and distances of the points for regional sampling of the upper left corner region, Assistant Alignment Pattern located on the left edge

Key

1	P_{UL}
2	P_1
3	P_2
4	P_3
5	L_x
6	$L_{x'}$
7	L_y
8	$L_{y'}$

Figure 32 — Symbol's top-left region

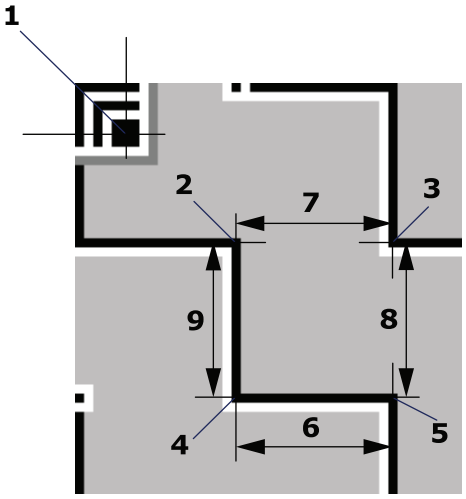
In the same way, find $L_{X'}$, which is the horizontal distance between the coordinates of P_{UL} and the coordinates of P_1, and $L_{Y'}$, which is the vertical distance between the coordinates of P_{UL} and the coordinates of P_2. Divide $L_{X'}$ and $L_{Y'}$ by the following formula to obtain the module pitches $CP_{X'}$ in the upper side and $CP_{Y'}$ in the left side in the upper left corner area of the symbol.

$CP_{X'} = L_{X'} / (AP$ - upper left Position Detection Center P_{UL}'s nominal y-ordinate)

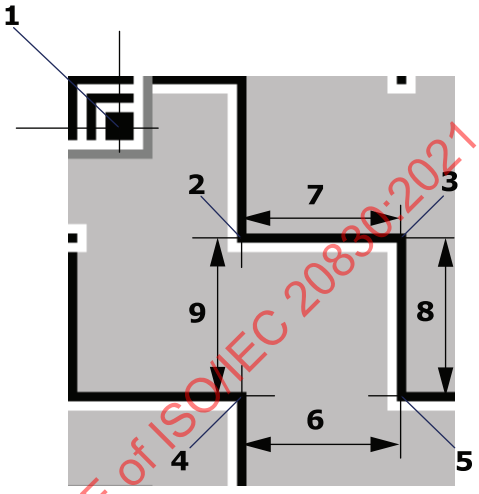
$CP_{Y'}=L_{Y'}/(AP$ - upper left Position Detection Center P_{UL}'s nominal x-ordinate)

g) Determine the sampling grid covering the upper left area of the symbol based on the module pitches CP_x, $CP_{X'}$, CP_y and $CP_{Y'}$ representing each side in the upper left area of the symbol.

Repeat steps a) to g) to determine the sampling grids for the upper right, lower right and lower left area of the symbol. The same principles shall be applied to determine sampling grid for any areas of the symbol, which are not sampled (see Figure 33 and Figure 34).



a) The relative positions and distances of the points for regional sampling of the regions besides the upper left corner region in the symbols of the first top-left Assistant Alignment Pattern located on the top edge

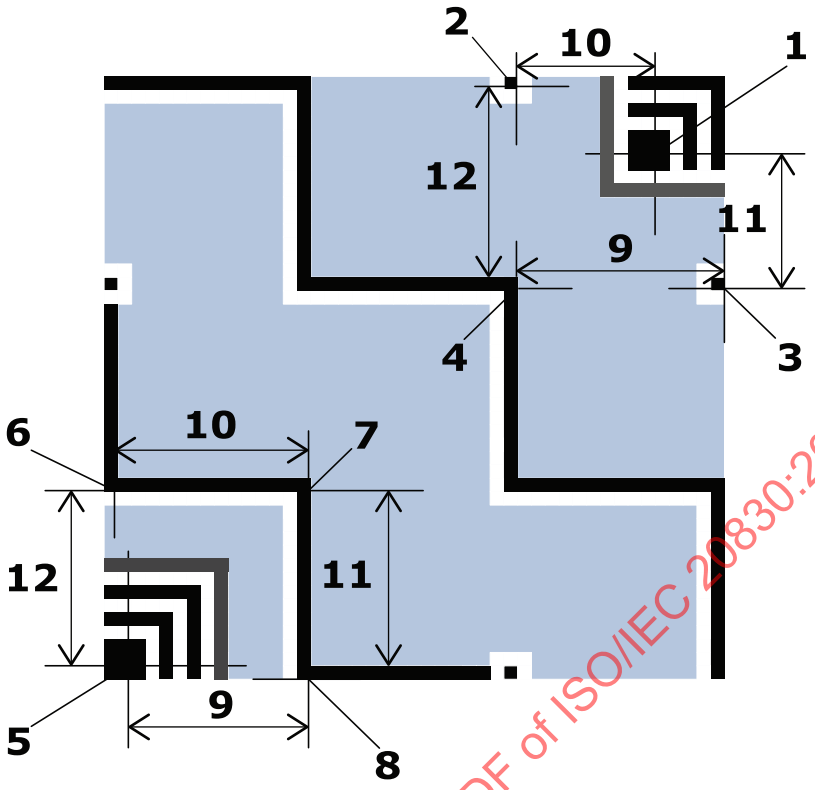


b) The relative positions and distances of the points for regional sampling of the regions besides the upper left corner region in the symbols of the first top-left Assistant Alignment Pattern located on the left edge

Key

1 P_{UL}
2 P_3
3 P_4
4 P_5
5 P_6
6 L_x
7 $L_{x'}$
8 L_y
9 $L_{y'}$

Figure 33 — The rest sampling region



Key

1	A
2	P_3
3	P_5
4	P_4
5	D
6	P_6
7	P_7
8	P_8
9	L_x
10	$L_{x'}$
11	L_y
12	$L_{y'}$

Figure 34 — The lower left and the upper right sampling region

10.6 Sampling

Sample image whether it is dark or light based on the global threshold. Construct a bit matrix mapping the dark modules as binary "1" and light modules as binary "0".

10.7 Masking releasing

According to masking solution obtained 10.4, XOR the data mask pattern with the encoding region of the symbol to release the data masking and restore the symbol characters representing data and error correction codewords. This is the reverse process of the data masking applied during the encoding procedure.

10.8 Restore data codewords

Restore the data codewords using the reverse of the process described in 5.8.3.

10.9 Error correction decoding

Follow the error detection and correction decoding procedure in 5.5 and Annex G to correct errors and erasures up to the maximum correction capacity for the symbol version and error correction level, and restore the original information codewords sequence. Erasures can be assumed, for example, when part of the symbol is perceived to be outside the field-of-view of the imager.

10.10 Data codeword decoding

Convert each codeword in the information codewords sequence received in 10.9.

Decode error correction to an 8-bit binary string and restore the information bit stream into the original data using the reverse of the process described in 5.4.

11 Transmitted data

11.1 General

This clause describes the standard transmission protocol for compliant readers. These readers may be programmable to support other transmission options.Once the structure of the data (including the use of any ECI) has been identified, the appropriate Symbology Identifier string which shall be compatible with the ISO/IEC 15424 should be added by the decoder as a preamble to the transmitted data; if ECIs are used the Symbology Identifier is required. See Annex L for the Symbology Identifier and option values which apply to Han Xin Code.

11.2 Basic interpretation

All encoded data characters are included in the data transmission. The symbology control characters and error correction characters are not transmitted.

11.3 Protocol for Extended Channel Interpretation

In systems where ECIs are supported, the use of a symbology identifier prefix is required with every transmission. 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 encoding. The next codeword(s) are converted into a 6-digit value, inverting the rules defined in Table 5. The 6-digit value is transmitted as the appropriate ASCII values (48-57).

Application software recognizing \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

Use of the symbology identifier assures that the application can correctly interpret the escape character.

11.4 Protocol for GS1 data transmission

In the GS1 related systems where GS1 mode Han Xin code are utilised, the use of a symbology identifier prefix (1000 0001) is required with every transmission, and the data group separator <FNC1> shall be transmitted as the $^{G}_{s}$ character $1D_{HEX}$.

Annex A
(normative)

Alignment Pattern parameters of symbol of different versions

Alignment Pattern parameters of Han Xin Code symbol of different versions are shown in Table A.1.

Table A.1 — Alignment Pattern parameters of symbol of different versions

Version	Size (module)	r (module)	k (module)	m
1	23×23	_	_	_
2	25×25	_	_	
3	27×27	_	_	_
4	29×29	15	_	1
5	31×31	15	_	1
6	33×33	17	_	1
7	35×35	18	_	1
8	37×37	19	_	1
9	39×39	20	_	1
10	41×41	21	_	1
11	43×43	15	14	2
12	45×45	15	15	2
13	47×47	15	16	2
14	49×49	17	16	2
15	51×51	17	17	2
16	53×53	19	17	2
17	55×55	19	18	2
18	57×57	19	19	2
19	59×59	19	20	2
20	61×61	21	20	2
21	63×63	21	21	2
22	65×65	17	16	3
23	67×67	16	17	3
24	69×69	18	17	3
25	71×71	17	18	3
26	73×73	19	18	3
27	75×75	18	19	3
28	77×77	20	19	3
29	79×79	19	20	3
30	81×81	21	20	3
31	83×83	20	21	3
32	85×85	17	17	4
33	87×87	19	17	4
34	89×89	17	18	4
35	91×91	19	18	4

Table A.1 *(continued)*

Version	Size (module)	r (module)	k (module)	m
36	93×93	17	19	4
37	95×95	19	19	4
38	97×97	21	19	4
39	99×99	19	20	4
40	101×101	21	20	4
41	103×103	18	17	5
42	105×105	20	17	5
43	107×107	17	18	5
44	109×109	19	18	5
45	111×111	21	18	5
46	113×113	18	19	5
47	115×115	20	19	5
48	117×117	22	19	5
49	119×119	17	17	6
50	121×121	19	17	6
51	123×123	15	18	6
52	125×125	17	18	6
53	127×127	19	18	6
54	129×129	21	18	6
55	131×131	17	19	6
56	133×133	19	19	6
57	135×135	21	19	6
58	137×137	18	17	7
59	139×139	20	17	7
60	141×141	15	18	7
61	143×143	17	18	7
62	145×145	19	18	7
63	147×147	21	18	7
64	149×149	16	19	7
65	151×151	18	19	7
66	153×153	17	17	8
67	155×155	19	17	8
68	157×157	21	17	8
69	159×159	15	18	8
70	161×161	17	18	8
71	163×163	19	18	8
72	165×165	21	18	8
73	167×167	15	19	8
74	169×169	17	19	8
75	171×171	18	17	9
76	173×173	20	17	9
77	175×175	22	17	9
78	177×177	15	18	9
79	179×179	17	18	9

Table A.1 *(continued)*

Version	Size (module)	r (module)	k (module)	m
80	181×181	19	18	9
81	183×183	21	18	9
82	185×185	23	18	9
83	187×187	17	17	10
84	189×189	19	17	10

Annex B
(normative)

Data capacity and error correction characterics of Han Xin Code

Data capacities and error correction characteristic of different versions of Han Xin Code are shown in Table B.1.

Table B.1 — Data capacity and error correction characteristics of Han Xin Code

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
1	L1	23	25	200	21	168	4	1	(25,21,4)
	L2				17	136	8	1	(25,17,8)
	L3				13	104	12	1	(25,13,12)
	L4				9	72	16	1	(25,9,16)
2	L1	25	37	296	31	248	6	1	(37,31,6)
	L2				25	200	12	1	(37,25,12)
	L3				19	152	18	1	(37,19,18)
	L4				15	120	22	1	(37,15,22)
3	L1	27	50	400	42	336	8	1	(50,42,8)
	L2				34	272	16	1	(50,34,16)
	L3				26	208	24	1	(50,26,24)
	L4				20	160	30	1	(50,20,30)
4	L1	29	54	432	46	368	8	1	(54,46,8)
	L2				38	304	16	1	(54,38,16)
	L3				30	240	24	1	(54,30,24)
	L4				22	176	32	1	(54,22,32)
5	L1	31	69	552	57	456	12	1	(69,57,12)
	L2				49	392	20	1	(69,49,20)
	L3				37	296	32	1	(69,37,32)
	L4				27	216	42	2	(34,14,20) (35,13,22)
6	L1	33	84	672	70	560	14	1	(84,70,14)
	L2				58	464	26	1	(84,58,26)
	L3				46	368	38	2	(44,24,20) (40,22,18)
	L4				34	272	50	2	(40,16,24) (44,18,26)
7	L1	35	100	800	84	672	16	1	(100,84,16)
	L2				70	560	30	1	(100,70,30)
	L3				54	432	46	2	(48,26,22) (52,28,24)
	L4				40	320	60	3	2(34,14,20) (32,12,20)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
8	L1	37	117	936	99	792	18	1	(117,99,18)
	L2				81	648	36	2	(58,40,18) (59,41,18)
	L3				63	504	54	2	(57,31,26) (60,32,28)
	L4				47	376	70	3	2(40,16,24) (37,15,22)
9	L1	39	136	1088	114	912	22	1	(136,114,22)
	L2				96	768	40	2	2(68,48,20)
	L3				74	592	62	3	2(44,24,20) (48,26,22)
	L4				54	432	82	3	2(46,18,28) (44,18,26)
10	L1	41	155	1240	131	1048	24	1	(155,131,24)
	L2				109	872	46	2	(74,52,22) (81,57,24)
	L3				83	664	72	3	2(51,27,24) (53,29,24)
	L4				61	488	94	3	2(53,21,32) (49,19,30)
11	L1	43	161	1288	135	1080	26	1	(161,135,26)
	L2				113	904	48	2	(80,56,24) (81,57,24)
	L3				87	696	74	3	2(52,28,24) (57,31,26)
	L4				65	520	96	3	2(54,22,32) (53,21,32)
12	L1	45	181	1448	153	1224	28	1	(181,153,28)
	L2				127	1016	54	2	(88,62,26) (93,65,28)
	L3				97	776	84	3	2(60,32,28) (61,33,28)
	L4				73	584	108	4	3(43,17,26) (52,22,30)
13	L1	47	203	1624	171	1368	32	2	(102,86,16) (101,85,16)
	L2				143	1144	60	2	(101,71,30) (102,72,30)
	L3				109	872	94	3	2(69,37,32) (65,35,30)
	L4				81	648	122	4	3(50,20,30) (53,21,32)
14	L1	49	225	1800	189	1512	36	2	(112,94,18) (113,95,18)
	L2				157	1256	68	3	2(73,51,22) (79,55,24)
	L3				121	968	104	4	3(56,30,26) (57,31,26)
	L4				89	712	136	5	4(46,18,28) (41,17,24)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
15	L1	51	249	1992	209	1672	40	2	(124,104,20) (125,105,20)
	L2				175	1400	74	3	2(81,57,24) (87,61,26)
	L3				135	1080	114	4	3(61,33,28) (66,36,30)
	L4				99	792	150	5	4(50,20,30) (49,19,30)
16	L1	53	273	2184	229	1832	44	2	(137,115,22) (136,114,22)
	L2				191	1528	82	3	2(93,65,28) (87,61,26)
	L3				147	1176	126	4	3(70,38,32) (63,33,30)
	L4				109	872	164	6	5(47,19,28) (38,14,24)
17	L1	55	299	2392	251	2008	48	2	(150,126,24) (149,125,24)
	L2				209	1672	90	3	2(100,70,30) (99,69,30)
	L3				161	1288	138	5	4(61,33,28) (55,29,26)
	L4				119	952	180	6	5(50,20,30) (49,19,30)
18	L1	57	325	2600	273	2184	52	2	(162,136,26) (163,137,26)
	L2				227	1816	98	4	3(80,56,24) (85,59,26)
	L3				175	1400	150	5	5(65,35,30)
	L4				129	1032	196	7	6(46,18,28) (49,21,28)
19	L1	59	353	2824	297	2376	56	2	(176,148,28) (177,149,28)
	L2				247	1976	106	4	3(87,61,26) (92,64,28)
	L3				191	1528	162	8	7(44,24,20) (45,23,22)
	L4				141	1128	212	7	6(50,20,30) (53,21,32)
20	L1	61	381	3048	321	2568	60	3	3(127,107,20)
	L2				267	2136	114	4	3(93,65,28) (102,72,30)
	L3				205	1640	176	8	7(48,26,22) (45,23,22)
	L4				153	1224	228	8	7(47,19,28) (52,20,32)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
21	L1	63	411	3288	345	2760	66	3	3(137,115,22)
	L2				287	2296	124	5	4(80,56,24) (91,63,28)
	L3				221	1768	190	8	7(52,28,24) (47,25,22)
	L4				165	1320	246	9	8(46,18,28) (43,21,22)
22	L1	65	422	3376	354	2832	68	3	2(138,116,22) (146,122,24)
	L2				296	2368	126	5	4(80,56,24) (102,72,30)
	L3				228	1824	194	8	7(52,28,24) (58,32,26)
	L4				168	1344	254	9	8(46,18,28) (54,24,30)
23	L1	67	453	3624	381	3048	72	3	3(151,127,24)
	L2				317	2536	136	6	5(73,51,22) (88,62,26)
	L3				245	1960	208	8	7(56,30,26) (61,35,26)
	L4				181	1448	272	9	8(50,20,30) (53,21,32)
24	L1	69	485	3880	407	3256	78	3	2(161,135,26) (163,137,26)
	L2				339	2712	146	6	5(80,56,24) (85,59,26)
	L3				261	2088	224	8	7(61,33,28) (58,30,28)
	L4				195	1560	290	12	11(40,16,24) (45,19,26)
25	L1	71	518	4144	436	3488	82	4	3(125,105,20) (143,121,22)
	L2				362	2896	156	6	5(87,61,26) (83,57,26)
	L3				280	2240	238	10	9(52,28,24) (50,28,22)
	L4				208	1664	310	11	10(47,19,28) (48,18,30)
26	L1	73	552	4416	464	3712	88	3	2(187,157,30) (178,150,28)
	L2				386	3088	166	6	5(93,65,28) (87,61,26)
	L3				298	2384	254	9	8(61,33,28) (64,34,30)
	L4				220	1760	332	12	10(47,19,28) 2(41,15,26)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
27	L1	75	587	4696	493	3944	94	4	3(150,126,24) (137,115,22)
	L2				411	3288	176	8	7(73,51,22) (76,54,22)
	L3				317	2536	270	9	8(65,35,30) (67,37,30)
	L4				235	1880	352	16	15(37,15,22) (32,10,22)
28	L1	77	623	4984	523	4184	100	5	4(125,105,20) (123,103,20)
	L2				437	3496	186	8	7(80,56,24) (63,45,18)
	L3				337	2696	286	11	10(57,31,26) (53,27,26)
	L4				251	2008	372	14	10(43,17,26) 3(48,20,28) (49,21,28)
29	L1	79	660	5280	554	4432	106	4	3(165,139,26) (165,137,28)
	L2				462	3696	198	7	6(94,66,28) (96,66,30)
	L3				358	2864	302	10	9(66,36,30) (66,34,32)
	L4				264	2112	396	14	13(47,19,28) (49,17,32)
30	L1	81	698	5584	586	4688	112	7	6(100,84,16) (98,82,16)
	L2				488	3904	210	7	6(100,70,30) (98,68,30)
	L3				376	3008	322	11	7(65,35,30) 3(61,33,28) (60,32,28)
	L4				280	2240	418	14	13(50,20,30) (48,20,28)
31	L1	83	737	5896	619	4952	118	6	5(125,105,20) (112,94,18)
	L2				515	4120	222	7	6(106,74,32) (101,71,30)
	L3				397	3176	340	12	11(61,33,28) (66,34,32)
	L4				295	2360	442	16	13(47,19,28) 3(42,16,26)
32	L1	85	754	6032	634	5072	120	5	4(151,127,24) (150,126,24)
	L2				528	4224	226	8	7(94,66,28) (96,66,30)
	L3				408	3264	346	14	12(54,30,24) (52,24,28) (54,24,30)
	L4				302	2416	452	16	15(47,19,28) (49,17,32)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
33	L1	87	794	6352	666	5328	128	8	7(100,84,16) (94,78,16)
	L2				556	4448	238	8	7(100,70,30) (94,66,28)
	L3				428	3424	366	13	12(61,33,28) (62,32,30)
	L4				318	2544	476	15	14(53,21,32) (52,24,28)
34	L1	89	836	6688	702	5616	134	6	5(139,117,22) (141,117,24)
	L2				586	4688	250	9	8(94,66,28) (84,58,26)
	L3				452	3616	384	12	11(70,38,32) (66,34,32)
	L4				334	2672	502	17	15(50,20,30) 2(43,17,26)
35	L1	91	878	7024	738	5904	140	5	4(176,148,28) (174,146,28)
	L2				614	4912	264	9	8(98,68,30) (94,70,24)
	L3				474	3792	404	13	10(68,36,32) 3(66,38,28)
	L4				352	2816	526	19	16(47,19,28) 3(42,16,26)
36	L1	93	922	7376	774	6192	148	6	4(150,126,24) 2(161,135,26)
	L2				646	5168	276	10	8(98,70,28) 2(69,43,26)
	L3				498	3984	424	15	13(60,32,28) 2(71,41,30)
	L4				368	2944	554	20	17(47,19,28) 3(41,15,26)
37	L1	95	966	7728	812	6496	154	6	5(162,136,26) (156,132,24)
	L2				676	5408	290	10	5(97,67,30) 4(96,68,28) (97,69,28)
	L3				522	4176	444	15	14(65,35,30) (56,32,24)
	L4				386	3088	580	22	18(44,18,26) 3(44,16,28) (42,14,28)
38	L1	97	1011	8088	849	6792	162	6	3(168,142,26) 3(169,141,28)
	L2				707	5656	304	10	8(100,70,30) (105,73,32) (106,74,32)
	L3				545	4360	466	16	12(64,34,30) 3(60,34,26) (63,35,28)
	L4				405	3240	606	19	18(53,21,32) (57,27,30)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
39	L1	99	1058	8464	888	7104	170	8	5(138,116,22) 2(123,103,20) (122,102,20)
	L2				740	5920	318	10	9(106,74,32) (104,74,30)
	L3				572	4576	486	17	14(62,34,28) 2(64,32,32) (62,32,30)
	L4				424	3392	634	20	19(53,21,32) (51,25,26)
40	L1	101	1105	8840	929	7432	176	8	7(138,116,22) (139,117,22)
	L2				773	6184	332	12	11(93,65,28) (82,58,24)
	L3				597	4776	508	16	15(70,38,32) (55,27,28)
	L4				441	3528	664	22	20(50,20,30) (52,20,32) (53,21,32)
41	L1	103	1126	9008	946	7568	180	7	6(162,136,26) (154,130,24)
	L2				788	6304	338	12	11(94,66,28) (92,62,30)
	L3				608	4864	518	18	14(62,34,28) 3(66,34,32) (60,30,30)
	L4				450	3600	676	23	18(50,20,30) 3(48,20,28) 2(41,15,26)
42	L1	105	1175	9400	987	7896	188	9	5(125,105,20) 2(137,115,22) 2(138,116,22)
	L2				823	6584	352	11	10(107,75,32) (105,73,32)
	L3				635	5080	540	17	16(70,38,32) (55,27,28)
	L4				469	3752	706	25	22(47,19,28) 2(46,16,30) (49,19,30)
43	L1	107	1224	9792	1028	8224	196	7	6(175,147,28) (174,146,28)
	L2				856	6848	368	13	11(94,66,28) 2(95,65,30)
	L3				660	5280	564	20	18(61,33,28) 2(63,33,30)
	L4				490	3920	734	23	22(53,21,32) (58,28,30)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
44	L1	109	1275	10200	1071	8568	204	9	6(138,116,22) 3(149,125,24)
	L2				893	7144	382	12	11(107,75,32) (98,68,30)
	L3				689	5512	586	20	13(63,35,28) 6(66,34,32) (60,30,30)
	L4				509	4072	766	24	23(53,21,32) (56,26,30)
45	L1	111	1327	10616	1115	8920	212	11	7(125,105,20) 4(113,95,18)
	L2				929	7432	398	14	12(95,67,28) (93,63,30) (94,62,32)
	L3				717	5736	610	23	21(57,31,26) 2(65,33,32)
	L4				531	4248	796	25	23(53,21,32) 2(54,24,30)
46	L1	113	1380	11040	1160	9280	220	10	10(138,116,22)
	L2				966	7728	414	13	12(106,74,32) (108,78,30)
	L3				746	5968	634	20	18(69,37,32) (69,39,30) (69,41,28)
	L4				552	4416	828	26	25(53,21,32) (55,27,28)
47	L1	115	1434	11472	1204	9632	230	10	5(150,126,24) 4(137,115,22) (136,114,22)
	L2				1004	8032	430	15	12(95,67,28) 2(98,66,32) (98,68,30)
	L3				774	6192	660	22	21(65,35,30) (69,39,30)
	L4				574	4592	860	27	26(53,21,32) (56,28,28)
48	L1	117	1489	11912	1251	10008	238	10	9(150,126,24) (139,117,22)
	L2				1043	8344	446	14	13(107,75,32) (98,68,30)
	L3				805	6440	684	23	20(65,35,30) 3(63,35,28)
	L4				595	4760	894	28	27(53,21,32) (58,28,30)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
49	L1	119	1513	12104	1271	10168	242	10	9(150,126,24) (163,137,26)
	L2				1059	8472	454	15	13(101,71,30) 2(100,68,32)
	L3				817	6536	696	22	20(69,37,32) (67,39,28) (66,38,28)
	L4				605	4840	908	29	24(52,20,32) 5(53,25,28)
50	L1	121	1569	12552	1317	10536	252	9	8(175,147,28) (169,141,28)
	L2				1099	8792	470	15	10(105,73,32) 4(104,74,30) (103,73,30)
	L3				847	6776	722	23	16(68,36,32) 6(69,39,30) (67,37,30)
	L4				627	5016	942	30	27(53,21,32) 3(46,20,26)
51	L1	123	1628	13024	1368	10944	260	10	9(163,137,26) (161,135,26)
	L2				1140	9120	488	16	12(100,70,30) 4(107,75,32)
	L3				880	7040	748	25	24(65,35,30) (68,40,28)
	L4				652	5216	976	31	23(52,20,32) 8(54,24,30)
52	L1	125	1686	13488	1416	11328	270	15	14(113,95,18) (104,86,18)
	L2				1180	9440	506	16	13(105,73,32) 3(107,77,30)
	L3				910	7280	776	26	24(65,35,30) 2(63,35,28)
	L4				674	5392	1012	32	26(53,21,32) 5(51,21,30) (53,23,30)
53	L1	127	1745	13960	1465	11720	280	10	9(175,147,28) (170,142,28)
	L2				1221	9768	524	17	10(103,73,30) 6(102,70,32) (103,71,32)
	L3				943	7544	802	27	25(65,35,30) 2(60,34,26)
	L4				697	5576	1048	33	29(53,21,32) 4(52,22,30)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
54	L1	129	1805	14440	1517	12136	288	12	11(150,126,24) (155,131,24)
	L2				1263	10104	542	17	16(106,74,32) (109,79,30)
	L3				975	7800	830	26	25(70,38,32) (55,25,30)
	L4				721	5768	1084	34	33(53,21,32) (56,28,28)
55	L1	131	1867	14936	1569	12552	298	15	14(125,105,20) (117,99,18)
	L2				1307	10456	560	20	19(93,65,28) (100,72,28)
	L3				1009	8072	858	27	24(69,37,32) 2(70,40,30) (71,41,30)
	L4				747	5976	1120	35	31(53,21,32) 4(56,24,32)
56	L1	133	1929	15432	1621	12968	308	11	10(175,147,28) (179,151,28)
	L2				1351	10808	578	19	15(101,71,30) 3(103,71,32) (105,73,32)
	L3				1041	8328	888	28	24(69,37,32) 3(68,38,30) (69,39,30)
	L4				771	6168	1158	39	36(49,19,30) 3(55,29,26)
57	L1	135	1992	15936	1674	13392	318	16	15(125,105,20) (117,99,18)
	L2				1394	11152	598	20	19(100,70,30) (92,64,28)
	L3				1076	8608	916	29	27(70,38,32) 2(51,25,26)
	L4				796	6368	1196	40	38(50,20,30) 2(46,18,28)
58	L1	137	2021	16168	1697	13576	324	16	14(125,105,20) (135,113,22) (136,114,22)
	L2				1415	11320	606	20	17(97,67,30) 3(124,92,32)
	L3				1091	8728	930	31	30(65,35,30) (71,41,30)
	L4				809	6472	1212	38	36(53,21,32) (56,26,30) (57,27,30)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
59	L1	139	2086	16688	1752	14016	334	12	11(174,146,28) (172,146,26)
	L2				1460	11680	626	21	20(100,70,30) (86,60,26)
	L3				1126	9008	960	30	29(70,38,32) (56,24,32)
	L4				834	6672	1252	42	40(50,20,30) 2(43,17,26)
60	L1	141	2151	17208	1807	14456	344	14	3(163,137,26) (162,136,26) 10(150,126,24)
	L2				1505	12040	646	23	22(93,65,28) (105,75,30)
	L3				1161	9288	990	31	30(69,37,32) (81,51,30)
	L4				861	6888	1290	43	42(50,20,30) (51,21,30)
61	L1	143	2218	17744	1864	14912	354	15	12(150,126,24) 2(140,118,22) (138,116,22)
	L2				1552	12416	666	21	19(106,74,32) (104,74,30) (100,72,28)
	L3				1198	9584	1020	32	30(70,38,32) 2(59,29,30)
	L4				892	7136	1326	42	39(52,20,32) 2(63,37,26) (64,38,26)
62	L1	145	2286	18288	1920	15360	366	15	12(150,126,24) 3(162,136,26)
	L2				1600	12800	686	23	21(100,70,30) 2(93,65,28)
	L3				1234	9872	1052	35	34(65,35,30) (76,44,32)
	L4				914	7312	1372	46	42(50,20,30) 2(47,19,28) 2(46,18,28)
63	L1	147	2355	18840	1979	15832	376	16	12(150,126,24) 3(139,117,22) (138,116,22)
	L2				1649	13192	706	27	25(87,61,26) 2(90,62,28)
	L3				1271	10168	1084	36	34(65,35,30) (72,40,32) (73,41,32)
	L4				941	7528	1414	47	45(50,20,30) (52,20,32) (53,21,32)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
64	L1	149	2425	19400	2037	16296	388	19	15(125,105,20) 2(137,115,22) 2(138,116,22)
	L2				1697	13576	728	26	25(93,65,28) (100,72,28)
	L3				1309	10472	1116	36	18(65,35,30) 17(69,37,32) (82,50,32)
	L4				969	7752	1456	49	42(50,20,30) 6(47,19,28) (43,15,28)
65	L1	151	2496	19968	2096	16768	400	20	19(125,105,20) (121,101,20)
	L2				1748	13984	748	34	33(73,51,22) (87,65,22)
	L3				1348	10784	1148	41	40(61,33,28) (56,28,28)
	L4				998	7984	1498	50	49(50,20,30) (46,18,28)
66	L1	153	2528	20224	2124	16992	404	20	18(125,105,20) 2(139,117,22)
	L2				1770	14160	758	27	26(93,65,28) (110,80,30)
	L3				1366	10928	1162	39	35(65,35,30) 3(63,35,28) (64,36,28)
	L4				1012	8096	1516	54	52(46,18,28) 2(68,38,30)
67	L1	155	2600	20800	2184	17472	416	26	26(100,84,16)
	L2				1820	14560	780	26	26(100,70,30)
	L3				1404	11232	1196	46	45(57,31,26) (35,9,26)
	L4				1040	8320	1560	52	52(50,20,30)
68	L1	157	2673	21384	2245	17960	428	18	16(150,126,24) (136,114,22) (137,115,22)
	L2				1871	14968	802	27	23(100,70,30) 3(93,65,28) (94,66,28)
	L3				1443	11544	1230	41	40(65,35,30) (73,43,30)
	L4				1069	8552	1604	54	46(50,20,30) 7(47,19,28) (44,16,28)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
69	L1	159	2749	21992	2309	18472	440	20	19(138,116,22) (127,105,22)
	L2				1925	15400	824	28	20(100,70,30) 7(94,66,28) (91,63,28)
	L3				1485	11880	1264	42	40(65,35,30) (74,42,32) (75,43,32)
	L4				1099	8792	1650	55	54(50,20,30) (49,19,30)
70	L1	161	2824	22592	2372	18976	452	19	17(150,126,24) 2(137,115,22)
	L2				1976	15808	848	28	24(100,70,30) 4(106,74,32)
	L3				1524	12192	1300	50	48(57,31,26) 2(44,18,26)
	L4				1130	9040	1694	61	54(47,19,28) 6(41,15,26) (40,14,26)
71	L1	163	2900	23200	2436	19488	464	29	29(100,84,16)
	L2				2030	16240	870	29	29(100,70,30)
	L3				1566	12528	1334	47	6(64,34,30) 3(66,36,30) 38(61,33,28)
	L4				1160	9280	1740	58	58(50,20,30)
72	L1	165	2977	23816	2501	20008	476	17	16(175,147,28) (177,149,28)
	L2				2083	16664	894	32	31(94,66,28) (63,37,26)
	L3				1607	12856	1370	49	48(61,33,28) (49,23,26)
	L4				1191	9528	1786	60	53(50,20,30) 6(47,19,28) (45,17,28)
73	L1	167	3056	24448	2568	20544	488	22	20(137,115,22) 2(158,134,24)
	L2				2140	17120	916	33	29(94,66,28) 2(82,56,26) 2(83,57,26)
	L3				1650	13200	1406	47	45(66,36,30) 2(43,15,28)
	L4				1222	9776	1834	61	59(50,20,30) 2(53,21,32)
74	L1	169	3135	25080	2633	21064	502	18	17(175,147,28) (160,134,26)
	L2				2195	17560	940	31	26(100,70,30) 5(107,75,32)
	L3				1693	13544	1442	48	47(65,35,30) (80,48,32)
	L4				1253	10024	1882	67	64(46,18,28) 2(63,33,30) (65,35,30)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
75	L1	171	3171	25368	2663	21304	508	23	22(137,115,22) (157,133,24)
	L2				2219	17752	952	34	33(93,65,28) (102,74,28)
	L3				1713	13704	1458	49	43(66,36,30) 5(55,27,28) (58,30,28)
	L4				1269	10152	1902	63	57(50,20,30) 5(53,21,32) (56,24,32)
76	L1	173	3252	26016	2732	21856	520	20	18(162,136,26) 2(168,142,26)
	L2				2276	18208	976	35	33(94,66,28) 2(75,49,26)
	L3				1756	14048	1496	50	48(65,35,30) 2(66,38,28)
	L4				1300	10400	1952	65	64(50,20,30) (52,20,32)
77	L1	175	3334	26672	2800	22400	534	22	19(150,126,24) 2(161,135,26) (162,136,26)
	L2				2334	18672	1000	36	32(94,66,28) 2(81,55,26) 2(82,56,26)
	L3				1800	14400	1534	51	49(66,36,30) 2(50,18,32)
	L4				1334	10672	2000	71	65(46,18,28) 5(57,27,30) (59,29,30)
78	L1	177	3416	27328	2870	22960	546	21	20(163,137,26) (156,130,26)
	L2				2392	19136	1024	32	30(107,75,32) 2(103,71,32)
	L3				1844	14752	1572	52	46(65,35,30) 6(71,39,32)
	L4				1366	10928	2050	73	3(42,12,30) 70(47,19,28)
79	L1	179	3500	28000	2940	23520	560	20	20(175,147,28)
	L2				2450	19600	1050	35	35(100,70,30)
	L3				1890	15120	1610	54	49(65,35,30) 5(63,35,28)
	L4				1400	11200	2100	70	70(50,20,30)
80	L1	181	3585	28680	3011	24088	574	22	21(162,136,26) (183,155,28)
	L2				2509	20072	1076	36	34(100,70,30) (92,64,28) (93,65,28)
	L3				1935	15480	1650	55	54(65,35,30) (75,45,30)
	L4				1433	11464	2152	72	68(50,20,30) 3(46,18,28) (47,19,28)

Table B.1 *(continued)*

Version	Error correction level	Number of modules on each side	Number of data codeword	Number of data bits	Number of information codeword	Number of information bits	Error correction codewords	Error correction blocks	Error correction code in Every block (n,k,2t)
81	L1	183	3671	29368	3083	24664	588	25	19(150,126,24) 5(137,115,22) (136,114,22)
	L2				2569	20552	1102	37	33(100,70,30) 3(93,65,28) (92,64,28)
	L3				1983	15864	1688	56	52(65,35,30) 3(73,41,32) (72,40,32)
	L4				1469	11752	2202	73	67(50,20,30) 5(53,21,32) (56,24,32)
82	L1	185	3758	30064	3156	25248	602	23	2(178,150,28) 21(162,136,26)
	L2				2630	21040	1128	38	32(100,70,30) 6(93,65,28)
	L3				2030	16240	1728	54	52(70,38,32) 2(59,27,32)
	L4				1504	12032	2254	75	73(50,20,30) 2(54,22,32)
83	L1	187	3798	30384	3190	25520	608	25	21(150,126,24) 4(162,136,26)
	L2				2658	21264	1140	36	30(106,74,32) 6(103,73,30)
	L3				2050	16400	1748	58	54(65,35,30) 4(72,40,32)
	L4				1520	12160	2278	76	75(50,20,30) (48,20,28)
84	L1	189	3886	31088	3264	26112	622	31	30(125,105,20) (136,114,22)
	L2				2720	21760	1166	58	3(67,45,22) 55(67,47,20)
	L3				2098	16784	1788	64	2(52,26,26) 62(61,33,28)
	L4				1554	12432	2332	83	79(46,18,28) 4(63,33,30)

Annex C
(informative)

Information capacity of Han Xin Code

Han Xin Code symbols of different versions and information capacity in different encoding modes mapping table as shown in Table C.1.

Table C.1 — Information capacity of Han Xin Code

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
1	L1	45	26	18	12	12	9	6
	L2	36	21	14	10	10	7	5
	L3	27	15	10	7	7	5	4
	L4	15	10	6	4	4	3	2
2	L1	69	39	28	19	19	15	9
	L2	54	31	22	15	15	12	8
	L3	39	23	16	11	11	8	6
	L4	30	18	12	8	8	6	4
3	L1	96	54	39	26	26	21	13
	L2	75	43	31	21	21	16	10
	L3	57	33	23	16	16	12	8
	L4	42	25	17	12	12	9	6
4	L1	105	59	43	29	29	23	14
	L2	87	49	35	24	24	19	12
	L3	66	38	27	18	18	14	9
	L4	48	27	19	13	13	10	7
5	L1	132	74	54	36	36	29	18
	L2	111	63	46	31	31	24	15
	L3	84	47	34	23	23	18	11
	L4	60	34	24	16	16	13	8
6	L1	162	91	67	45	45	36	22
	L2	135	75	55	37	37	29	18
	L3	105	59	43	29	29	23	14
	L4	75	43	31	21	21	16	10
7	L1	195	110	81	54	54	43	26
	L2	162	91	67	45	45	36	22
	L3	123	70	51	34	34	27	17
	L4	90	51	37	25	25	20	12

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
8	L1	231	130	96	64	64	51	31
	L2	189	106	78	52	52	41	25
	L3	147	82	60	40	40	32	20
	L4	108	61	44	30	30	23	15
9	L1	267	150	111	74	74	59	36
	L2	225	126	93	62	62	49	30
	L3	171	97	71	48	48	38	23
	L4	123	70	51	34	34	27	17
10	L1	309	173	128	86	86	68	41
	L2	255	143	106	71	71	56	34
	L3	195	109	80	54	54	43	26
	L4	141	79	58	39	39	31	19
11	L1	318	178	132	88	88	70	43
	L2	267	149	110	74	74	59	36
	L3	204	114	84	56	56	45	27
	L4	150	85	62	42	42	33	20
12	L1	363	202	150	100	100	80	48
	L2	300	167	124	83	83	66	40
	L3	228	127	94	63	63	50	31
	L4	171	95	70	47	47	37	23
13	L1	405	226	168	112	112	89	54
	L2	339	189	140	94	94	75	45
	L3	255	143	106	71	71	56	34
	L4	189	106	78	52	52	41	25
14	L1	447	250	186	124	124	99	60
	L2	372	207	154	103	103	82	50
	L3	285	159	118	79	79	63	38
	L4	207	117	86	58	58	46	28
15	L1	495	277	206	138	138	110	66
	L2	414	231	172	115	115	92	56
	L3	318	178	132	88	88	70	43
	L4	231	130	96	64	64	51	31
16	L1	543	303	226	151	151	120	73
	L2	453	253	188	126	126	100	61
	L3	348	194	144	96	96	77	47
	L4	255	143	106	71	71	56	34
17	L1	597	333	248	166	166	132	80
	L2	495	277	206	138	138	110	66
	L3	381	213	158	106	106	84	51
	L4	279	157	116	78	78	62	38

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
18	L1	651	362	270	180	180	144	87
	L2	540	301	224	150	150	119	72
	L3	414	231	172	115	115	92	56
	L4	303	170	126	84	84	67	41
19	L1	708	394	294	196	196	157	95
	L2	588	327	244	163	163	130	79
	L3	453	253	188	126	126	100	61
	L4	333	186	138	92	92	73	45
20	L1	765	426	318	212	212	169	102
	L2	636	354	264	176	176	141	85
	L3	486	271	202	135	135	108	65
	L4	363	202	150	100	100	80	48
21	L1	822	458	342	228	228	182	110
	L2	684	381	284	190	190	151	91
	L3	525	293	218	146	146	116	70
	L4	390	218	162	108	108	86	52
22	L1	843	470	351	234	234	187	113
	L2	705	393	293	196	196	156	94
	L3	543	302	225	150	150	120	72
	L4	399	222	165	110	110	88	53
23	L1	909	506	378	252	252	201	121
	L2	756	421	314	210	210	167	101
	L3	582	325	242	162	162	129	78
	L4	429	239	178	119	119	95	57
24	L1	972	541	404	270	270	215	130
	L2	807	450	336	224	224	179	108
	L3	621	346	258	172	172	137	83
	L4	462	258	192	128	128	102	62
25	L1	1041	579	433	289	289	231	139
	L2	864	481	359	240	240	191	115
	L3	666	371	277	185	185	148	89
	L4	495	275	205	137	137	109	66
26	L1	1107	617	461	308	308	246	148
	L2	921	513	383	256	256	204	123
	L3	711	395	295	197	197	157	95
	L4	522	291	217	145	145	116	70
27	L1	1179	655	490	327	327	261	157
	L2	981	546	408	272	272	217	131
	L3	756	421	314	210	210	167	101
	L4	558	311	232	155	155	124	75

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
28	L1	1251	695	520	347	347	277	167
	L2	1044	581	434	290	290	231	139
	L3	804	447	334	223	223	178	107
	L4	597	333	248	166	166	132	80
29	L1	1323	737	551	368	368	294	177
	L2	1104	614	459	306	306	245	147
	L3	855	475	355	237	237	189	114
	L4	627	350	261	174	174	139	84
30	L1	1401	779	583	389	389	311	187
	L2	1167	649	485	324	324	259	156
	L3	897	499	373	249	249	199	120
	L4	666	371	277	185	185	148	89
31	L1	1479	823	616	411	411	328	198
	L2	1230	685	512	342	342	273	164
	L3	948	527	394	263	263	210	127
	L4	702	391	292	195	195	156	94
32	L1	1515	843	631	421	421	336	202
	L2	1263	702	525	350	350	280	168
	L3	975	542	405	270	270	216	130
	L4	720	401	299	200	200	159	96
33	L1	1593	886	663	442	442	353	213
	L2	1329	739	553	369	369	295	177
	L3	1023	569	425	284	284	227	136
	L4	759	422	315	210	210	168	101
34	L1	1680	934	699	466	466	373	224
	L2	1401	779	583	389	389	311	187
	L3	1080	601	449	300	300	239	144
	L4	795	443	331	221	221	176	106
35	L1	1767	982	735	490	490	392	236
	L2	1467	817	611	408	408	326	196
	L3	1131	630	471	314	314	251	151
	L4	840	467	349	233	233	186	112
36	L1	1851	1030	771	514	514	411	247
	L2	1545	859	643	429	429	343	206
	L3	1191	662	495	330	330	264	159
	L4	879	489	365	244	244	195	117
37	L1	1944	1081	809	540	540	431	259
	L2	1617	899	673	449	449	359	216
	L3	1248	694	519	346	346	277	167
	L4	921	513	383	256	256	204	123

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
38	L1	2031	1130	846	564	564	451	271
	L2	1692	941	704	470	470	375	226
	L3	1302	725	542	362	362	289	174
	L4	966	538	402	268	268	214	129
39	L1	2127	1182	885	590	590	472	284
	L2	1770	985	737	492	492	393	236
	L3	1368	761	569	380	380	303	183
	L4	1011	563	421	281	281	224	135
40	L1	2223	1237	926	618	618	494	297
	L2	1851	1029	770	514	514	411	247
	L3	1428	794	594	396	396	317	191
	L4	1053	586	438	292	292	233	141
41	L1	2265	1259	943	629	629	503	302
	L2	1887	1049	785	524	524	419	252
	L3	1455	809	605	404	404	323	194
	L4	1074	598	447	298	298	238	144
42	L1	2364	1314	984	656	656	525	315
	L2	1971	1095	820	547	547	437	263
	L3	1518	845	632	422	422	337	203
	L4	1119	623	466	311	311	248	150
43	L1	2463	1369	1025	684	684	547	328
	L2	2049	1139	853	569	569	455	273
	L3	1578	878	657	438	438	350	211
	L4	1170	651	487	325	325	260	156
44	L1	2565	1426	1068	712	712	569	342
	L2	2139	1189	890	594	594	475	285
	L3	1647	917	686	458	458	366	220
	L4	1215	677	506	338	338	270	162
45	L1	2670	1485	1112	742	742	593	356
	L2	2223	1237	926	618	618	494	297
	L3	1716	954	714	476	476	381	229
	L4	1269	706	528	352	352	281	169
46	L1	2778	1545	1157	772	772	617	371
	L2	2313	1286	963	642	642	513	309
	L3	1785	993	743	496	496	396	238
	L4	1320	734	549	366	366	293	176
47	L1	2883	1603	1201	801	801	640	385
	L2	2403	1337	1001	668	668	534	321
	L3	1851	1030	771	514	514	411	247
	L4	1371	763	571	381	381	304	183

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
48	L1	2997	1666	1248	832	832	665	400
	L2	2499	1389	1040	694	694	555	333
	L3	1926	1071	802	535	535	428	257
	L4	1422	791	592	395	395	316	190
49	L1	3045	1693	1268	846	846	676	406
	L2	2535	1410	1056	704	704	563	338
	L3	1956	1087	814	543	543	434	261
	L4	1446	805	602	402	402	321	193
50	L1	3156	1754	1314	876	876	701	421
	L2	2631	1463	1096	731	731	584	351
	L3	2028	1127	844	563	563	450	271
	L4	1500	834	624	416	416	333	200
51	L1	3279	1822	1365	910	910	728	437
	L2	2730	1518	1137	758	758	606	364
	L3	2106	1171	877	585	585	468	281
	L4	1560	867	649	433	433	346	208
52	L1	3393	1886	1413	942	942	753	453
	L2	2826	1571	1177	785	785	628	377
	L3	2178	1211	907	605	605	484	291
	L4	1611	897	671	448	448	358	215
53	L1	3510	1951	1462	975	975	780	468
	L2	2925	1626	1218	812	812	649	390
	L3	2259	1255	940	627	627	501	301
	L4	1668	927	694	463	463	370	223
54	L1	3636	2021	1514	1010	1010	807	485
	L2	3027	1682	1260	840	840	672	404
	L3	2334	1298	972	648	648	518	312
	L4	1725	959	718	479	479	383	230
55	L1	3759	2090	1566	1044	1044	835	502
	L2	3132	1741	1304	870	870	695	418
	L3	2415	1343	1006	671	671	536	322
	L4	1788	994	744	496	496	397	239
56	L1	3885	2159	1618	1079	1079	863	518
	L2	3237	1799	1348	899	899	719	432
	L3	2493	1386	1038	692	692	553	333
	L4	1845	1026	768	512	512	409	246
57	L1	4011	2230	1671	1114	1114	891	535
	L2	3339	1857	1391	928	928	742	446
	L3	2577	1433	1073	716	716	572	344
	L4	1905	1059	793	529	529	423	254

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
58	L1	4068	2261	1694	1130	1130	903	543
	L2	3390	1885	1412	942	942	753	452
	L3	2613	1453	1088	726	726	580	349
	L4	1935	1077	806	538	538	430	258
59	L1	4200	2334	1749	1166	1166	933	560
	L2	3498	1945	1457	972	972	777	467
	L3	2697	1499	1123	749	749	599	360
	L4	1995	1110	831	554	554	443	266
60	L1	4332	2407	1804	1203	1203	962	578
	L2	3606	2005	1502	1002	1002	801	481
	L3	2781	1546	1158	772	772	617	371
	L4	2061	1146	858	572	572	457	275
61	L1	4467	2483	1861	1241	1241	992	596
	L2	3720	2067	1549	1033	1033	826	496
	L3	2871	1595	1195	797	797	637	383
	L4	2136	1187	889	593	593	474	285
62	L1	4602	2558	1917	1278	1278	1022	614
	L2	3834	2131	1597	1065	1065	852	512
	L3	2955	1643	1231	821	821	656	394
	L4	2187	1217	911	608	608	486	292
63	L1	4743	2637	1976	1318	1318	1054	633
	L2	3951	2197	1646	1098	1098	878	527
	L3	3045	1693	1268	846	846	676	406
	L4	2253	1253	938	626	626	500	301
64	L1	4884	2714	2034	1356	1356	1085	651
	L2	4068	2261	1694	1130	1130	903	543
	L3	3135	1743	1306	871	871	696	418
	L4	2319	1290	966	644	644	515	310
65	L1	5025	2793	2093	1396	1396	1116	670
	L2	4191	2329	1745	1164	1164	931	559
	L3	3231	1795	1345	897	897	717	431
	L4	2391	1329	995	664	664	531	319
66	L1	5091	2830	2121	1414	1414	1131	679
	L2	4242	2358	1767	1178	1178	942	566
	L3	3273	1819	1363	909	909	727	437
	L4	2424	1347	1009	673	673	538	323
67	L1	5235	2910	2181	1454	1454	1163	698
	L2	4362	2425	1817	1212	1212	969	582
	L3	3363	1870	1401	934	934	747	449
	L4	2490	1385	1037	692	692	553	332

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
68	L1	5382	2991	2242	1495	1495	1196	718
	L2	4485	2493	1868	1246	1246	996	598
	L3	3459	1922	1440	960	960	768	461
	L4	2559	1423	1066	711	711	568	342
69	L1	5535	3077	2306	1538	1538	1230	738
	L2	4614	2565	1922	1282	1282	1025	616
	L3	3558	1978	1482	988	988	790	475
	L4	2631	1463	1096	731	731	584	351
70	L1	5688	3161	2369	1580	1580	1263	759
	L2	4737	2633	1973	1316	1316	1052	632
	L3	3651	2030	1521	1014	1014	811	487
	L4	2706	1505	1127	752	752	601	361
71	L1	5841	3246	2433	1622	1622	1297	779
	L2	4866	2705	2027	1352	1352	1081	649
	L3	3753	2086	1563	1042	1042	833	501
	L4	2778	1545	1157	772	772	617	371
72	L1	5997	3333	2498	1666	1666	1332	800
	L2	4995	2775	2080	1387	1387	1109	666
	L3	3852	2141	1604	1070	1070	855	514
	L4	2853	1586	1188	792	792	633	381
73	L1	6159	3422	2565	1710	1710	1368	821
	L2	5130	2851	2137	1425	1425	1140	684
	L3	3954	2198	1647	1098	1098	878	528
	L4	2928	1627	1219	813	813	650	391
74	L1	6315	3509	2630	1754	1754	1403	842
	L2	5262	2925	2192	1462	1462	1169	702
	L3	4059	2255	1690	1127	1127	901	541
	L4	3003	1669	1250	834	834	667	400
75	L1	6387	3549	2660	1774	1774	1419	852
	L2	5319	2957	2216	1478	1478	1182	710
	L3	4107	2282	1710	1140	1140	912	548
	L4	3039	1690	1266	844	844	675	406
76	L1	6552	3641	2729	1820	1820	1455	874
	L2	5457	3033	2273	1516	1516	1212	728
	L3	4209	2339	1753	1169	1169	935	561
	L4	3114	1731	1297	865	865	692	416
77	L1	6714	3731	2797	1865	1865	1492	896
	L2	5595	3110	2331	1554	1554	1243	746
	L3	4314	2398	1797	1198	1198	958	576
	L4	3195	1777	1331	888	888	710	426

Table C.1 *(continued)*

Version	Error correction level	Data capacity						
		Numeric mode	Text mode	Binary byte mode	Common Chinese Characters in Region One mode	Common Chinese Characters in Region Two mode	GB18030 2-byte Region mode	GB18030 4-byte Region mode
78	L1	6882	3825	2867	1912	1912	1529	918
	L2	5736	3187	2389	1593	1593	1274	765
	L3	4419	2457	1841	1228	1228	982	590
	L4	3273	1819	1363	909	909	727	437
79	L1	7050	3918	2937	1958	1958	1566	940
	L2	5874	3265	2447	1632	1632	1305	784
	L3	4530	2518	1887	1258	1258	1006	604
	L4	3354	1865	1397	932	932	745	448
80	L1	7221	4013	3008	2006	2006	1604	963
	L2	6015	3343	2506	1671	1671	1336	802
	L3	4638	2578	1932	1288	1288	1030	619
	L4	3435	1909	1430	954	954	763	458
81	L1	7395	4109	3080	2054	2054	1643	986
	L2	6159	3423	2566	1711	1711	1368	822
	L3	4755	2642	1980	1320	1320	1056	634
	L4	3519	1957	1466	978	978	782	470
82	L1	7569	4206	3153	2102	2102	1681	1009
	L2	6306	3505	2627	1752	1752	1401	841
	L3	4866	2705	2027	1352	1352	1081	649
	L4	3603	2003	1501	1001	1001	800	481
83	L1	7650	4251	3187	2125	2125	1700	1020
	L2	6375	3542	2655	1770	1770	1416	850
	L3	4914	2731	2047	1365	1365	1092	656
	L4	3642	2025	1517	1012	1012	809	486
84	L1	7827	4350	3261	2174	2174	1739	1044
	L2	6522	3625	2717	1812	1812	1449	870
	L3	5031	2795	2095	1397	1397	1117	671
	L4	3723	2070	1551	1034	1034	827	497

Annex D
(normative)

Error correction codeword generator polynomials

Table D.1 lists are error correction codeword generator polynomials. where α is the generator of GF(2^8); and every generator polynomial is the product of one-degree polynomial $x-\alpha^{0}$, $x-\alpha^{1}$, ..., $x-\alpha^{n}$, where n is the degree of generator polynomial.

Table D.1 — Error correction codeword generator polynomials

Number of error correction codeword	Generator polynomial
2	$x^2+\alpha^{198}x+\alpha^3$
4	$x^4+\alpha^{82}x^3+\alpha^{250}x^2+\alpha^{87}x+\alpha^{10}$
6	$x^6+\alpha^{159}x^5+\alpha^{88}x^4+\alpha^{64}x^3+\alpha^{95}x^2+\alpha^{173}x+\alpha^{21}$
8	$x^8+\alpha^{105}x^7+\alpha^{139}x^6+\alpha^{192}x^5+\alpha^{239}x^4+\alpha^{201}x^3+\alpha^{157}x^2+\alpha^{132}x+\alpha^{36}$
10	$x^{10}+\alpha^{191}x^9+\alpha^{74}x^8+\alpha^{73}x^7+\alpha^{225}x^6+\alpha^{9}x^5+\alpha^{236}x^4+\alpha^{95}x^3+\alpha^{107}x^2+\alpha^{235}x+\alpha^{55}$
12	$x^{12}+\alpha^{4}x^{11}+\alpha^{113}x^{10}+\alpha^{198}x^9+\alpha^{199}x^8+\alpha^{184}x^7+\alpha^{6}x^6+\alpha^{197}x^5+$ $\alpha^{225}x^4+\alpha^{237}x^3+\alpha^{165}x^2+\alpha^{69}x+\alpha^{78}$
14	$x^{14}+\alpha^{146}x^{13}+\alpha^{229}x^{12}+\alpha^{127}x^{11}+\alpha^{99}x^{10}+\alpha^{170}x^9+\alpha^{96}x^8+\alpha^{233}x^7+$ $\alpha^{111}x^6+\alpha^{200}x^5+\alpha^{144}x^4+\alpha^{187}x^3+\alpha^{49}x^2+\alpha^{236}x+\alpha^{105}$
16	$x^{16}+\alpha^{151}x^{15}+\alpha^{183}x^{14}+\alpha^{223}x^{13}+\alpha^{169}x^{12}+\alpha^{53}x^{11}+\alpha^{205}x^{10}+\alpha^{173}x^9+\alpha^{155}x^8+\alpha^{190}x^7+$ $\alpha^{239}x^6+\alpha^{104}x^5+\alpha^{237}x^4+\alpha^{53}x^3+\alpha^{30}x^2+\alpha^{15}x+\alpha^{136}$
18	$x^{18}+\alpha^{99}x^{17}+\alpha^{30}x^{16}+\alpha^{75}x^{15}+\alpha^{225}x^{14}+\alpha^{251}x^{13}+\alpha^{122}x^{12}+\alpha^{158}x^{11}+\alpha^{111}x^{10}+$ $\alpha^{232}x^9+\alpha^{130}x^8+\alpha^{196}x^7+\alpha^{179}x^6+\alpha^{72}x^5+\alpha^{65}x^4+\alpha^{189}x^3+\alpha^{163}x^2+\alpha^{251}x+\alpha^{171}$
20	$x^{20}+\alpha^{68}x^{19}+\alpha^{182}x^{18}+\alpha^{175}x^{17}+\alpha^{224}x^{16}+\alpha^{75}x^{14}+\alpha^{253}x^{13}+\alpha^{180}x^{12}+\alpha^{114}x^{11}+\alpha^{238}x^{10}+$ $\alpha^{135}x^9+\alpha^{222}x^8+\alpha^{61}x^7+\alpha^{159}x^6+\alpha^{105}x^5+\alpha^{95}x^4+\alpha^{67}x^3+\alpha^{95}x^2+\alpha^{2}x+\alpha^{210}$
22	$x^{22}+\alpha^{41}x^{21}+\alpha^{118}x^{20}+\alpha^{80}x^{19}+\alpha^{57}x^{18}+\alpha^{36}x^{17}+\alpha^{10}x^{16}+\alpha^{193}x^{15}+\alpha^{69}x^{14}+\alpha^{145}x^{13}+$ $\alpha^{243}x^{12}+\alpha^{208}x^{11}+\alpha^{11}x^{10}+\alpha^{191}x^9+\alpha^{138}x^8+\alpha^{30}x^7+\alpha^{125}x^6+\alpha^{174}x^5+\alpha^{218}x^4+\alpha^{9}x^3+$ $\alpha^{70}x^2+\alpha^{16}x+\alpha^{253}$
24	$x^{24}+\alpha^{204}x^{23}+\alpha^{71}x^{22}+\alpha^{6}x^{21}+\alpha^{201}x^{20}+\alpha^{149}x^{19}+\alpha^{51}x^{18}+\alpha^{182}x^{17}+\alpha^{212}x^{16}+\alpha^{38}x^{15}+$ $\alpha^{85}x^{14}+\alpha^{192}x^{13}+\alpha^{224}x^{12}+\alpha^{217}x^{11}+\alpha^{135}x^{10}+\alpha^{113}x^{9}+\alpha^{57}x^8+\alpha^{52}x^7+\alpha^{201}x^6+$ $\alpha^{69}x^5+\alpha^{146}x^4+\alpha^{231}x^3+\alpha^{66}x^2+\alpha^{224}x+\alpha^{45}$
26	$x^{26}+\alpha^{244}x^{25}+\alpha^{236}x^{24}+\alpha^{79}x^{23}+\alpha^{64}x^{22}+\alpha^{240}x^{21}+\alpha^{105}x^{20}+\alpha^{205}x^{19}+\alpha^{163}x^{18}+\alpha^{192}x^{17}+$ $\alpha^{138}x^{16}+\alpha^{250}x^{15}+\alpha^{231}x^{14}+\alpha^{111}x^{13}+\alpha^{3}x^{12}+\alpha^{49}x^{11}+\alpha^{219}x^{10}+\alpha^{45}x^9+\alpha^{43}x^8+\alpha^{112}x^7+$ $\alpha^{39}x^6+\alpha^{201}x^5+\alpha^{52}x^4+\alpha^{94}x^3+\alpha^{23}x^2+\alpha^{58}x+\alpha^{96}$
28	$x^{28}+\alpha^{233}x^{27}+\alpha^{18}x^{26}+\alpha^{156}x^{25}+\alpha^{11}x^{24}+\alpha^{95}x^{23}+\alpha^{5}x^{22}+\alpha^{78}x^{21}+\alpha^{254}x^{20}+\alpha^{252}x^{19}+$ $\alpha^{126}x^{18}+\alpha^{186}x^{17}+\alpha^{66}x^{16}+\alpha^{206}x^{15}+\alpha^{47}x^{14}+\alpha^{235}x^{13}+\alpha^{124}x^{12}+\alpha^{18}x^{11}+\alpha^{242}x^{10}+\alpha^{142}x^9+$ $\alpha^{173}x^8+\alpha^{26}x^7+\alpha^{237}x^6+\alpha^{101}x^5+\alpha^{46}x^4+\alpha^{220}x^3+\alpha^{111}x^2+\alpha^{100}x+\alpha^{151}$

Table D.1 *(continued)*

Number of error correction codeword	Generator polynomial
30	$x^{30}+\alpha^{142}x^{29}+\alpha^{162}x^{28}+\alpha^{34}x^{27}+\alpha^{192}x^{26}+\alpha^{61}x^{25}+\alpha^{96}x^{24}+\alpha^{77}x^{23}+\alpha^{43}x^{22}+\alpha^{14}x^{21}+$ $\alpha^{106}x^{20}+\alpha^{135}x^{19}+\alpha^{198}x^{18}+\alpha^{31}x^{17}+\alpha^{26}x^{16}+\alpha^{219}x^{15}+\alpha^{57}x^{14}+\alpha^{93}x^{13}+\alpha^{36}x^{12}+\alpha^{4}x^{11}+$ $\alpha^{6}x^{10}+\alpha^{200}x^{9}+\alpha^{5}x^{8}+\alpha^{70}x^{7}+\alpha^{120}x^{6}+\alpha^{116}x^{5}+\alpha^{23}x^{4}+\alpha^{151}x^{3}+\alpha^{55}x^{2}+\alpha^{66}x+\alpha^{210}$
32	$x^{32}+\alpha^{243}x^{31}+\alpha^{254}x^{30}+\alpha^{35}x^{29}+\alpha^{173}x^{28}+\alpha^{31}x^{27}+\alpha^{114}x^{26}+\alpha^{135}x^{25}+\alpha^{226}x^{24}+\alpha^{105}x^{23}+$ $\alpha^{242}x^{22}+\alpha^{244}x^{21}+\alpha^{15}x^{20}+\alpha^{72}x^{19}+\alpha^{250}x^{18}+\alpha^{136}x^{17}+\alpha^{128}x^{16}+\alpha^{169}x^{15}+$ $\alpha^{61}x^{14}+\alpha^{171}x^{13}+\alpha^{147}x^{12}+\alpha^{154}x^{11}+\alpha^{185}x^{10}+\alpha^{81}x^{9}+\alpha^{235}x^{8}+\alpha^{177}x^{7}+$ $\alpha^{189}x^{6}+\alpha^{139}x^{5}+\alpha^{59}x^{4}+\alpha^{209}x^{3}+\alpha^{206}x^{2}+\alpha^{228}x+\alpha^{18}$

Annex E
(normative)

Structural Information

E.1 General

Structural Information consists of 8 bits of version information, 2 bits of error correction level information, 2 bits of masking to make up of 12 bits of data. Error correction of Structural Information adopts Reed-Solomon error correction code of the GF(2^4) (see Figure E.1). This Annex specifies the calculation methods of the error correction bit of Structural Information and the process of the error correction decoding.

$$\underbrace{X_1\ X_2\ X_3\ X_4\ X_5\ X_6\ X_7\ X_8}_{\mathbf{1}}\ \underbrace{X_9\ X_{10}}_{\mathbf{2}}\ \underbrace{X_{11}\ X_{12}}_{\mathbf{3}}$$

Key

1 Version+20

2 Error correction level

3 Mask index

Figure E.1 — Structure of function information

E.2 Error correction encoding of Structural Information

Step 1: Divide the 12 bits of bit stream into 3 codewords (the length of each codeword is 4 bits).

$$X_1X_2X_3X_4\ X_5X_6X_7X_8\ X_9X_{10}X_{11}X_{12}$$

Step 2: Construct the error correction code (c, k, $2t$), k=3 is the result of the step 1, choose t as 2, then c=7 can be obtained, i.e. error correction code is (7,3,4).

Step 3: Construct information polynomial: $m(x)=m_2x^2+m_1x+m_0$, where: coefficient m_2、m_1、m_0 of multinomial are the values of those 3 codeword respectively.

Step 4: Construct information polynomial:

$$g(x)=(x+\alpha)\left(x+\alpha^2\right)\left(x+\alpha^3\right)\left(x+\alpha^4\right)=x^4+\beta_3x^3+\beta_2x^2+\beta_1x+\beta_0$$

where α is the generator on GF(2^4), and $\alpha^4+\alpha+1=0$

Step 5: Generate the error correction code. Error correction code is the information polynomial multiply X^{2t} to generate a multinomial, then divide the residue, i.e. $r_3x^3+r_2x^2+r_1x+r_0$, therefore displacement is: (r_3, r_2, r_1, r_0).(where, r_3, r_2, r_1, r_0 are all with the length of 4 code elements)

Step 6: The bit stream of the terminal Structural Information is (m_2,m_1,m_0,r_3,r_2,r_1,r_0).

For example, the Structural Information of a code pattern symbol is shown in Table E.1.

Table E.1 — Structural Information of symbol

Function information types	Function information values	Encoding
Version information	50+20	01000110
Error correction level	L1 (8%)	00
Masking solution	((ij)mod 2+(ij)mod 3) mod 2=0	10

Bit stream of Structural Information:010001100010

Construct polynomial:m(x)=$4x^2$+6x+2=$\alpha^2x^2+\alpha^5x+\alpha$

Construct generator polynomial:g(x)= $x^4+\alpha^{13}x^3+\alpha^6x^2+\alpha^3x+\alpha^{10}$

Error correction codeword:m(x)× x^4 =$\alpha^2x^6+\alpha^5x^5+\alpha x^4$

Calculate ($\alpha^2x^6+\alpha^5x^5+\alpha x^4$) mod ($x^4+\alpha^{13}x^3+\alpha^6x^2+\alpha^3x+\alpha^{10}$)

Error correction codeword obtained:1101 1001 0101 1110

The final Structural Information bit stream acquired by adding error correction is:0100 0110 0010 1101 1001 0101 1110

E.3 Error correction decoding of Structural Information

Supposing the following codes from Structural Information region of a symbol:

R = (r_6,r_5, ,r_1,r_0)

R(X) = $r_6x^6+r_5x^5+ +r_1x+r_0$

r_i(i=0,K,6)is the element of GF(2^4)field.

1. Calculate syndrome polynomial.

$$S_1=R(\alpha)=r_6\alpha^6+r_5\alpha^5+\cdots+r_1\alpha+r_0$$
$$S_2=R(\alpha^2)=r_6\alpha^{12}+r_5\alpha^{10}+\cdots+r_1\alpha^2+r_0$$
$$S_3=R(\alpha^3)=r_6\alpha^{18}+r_5\alpha^{15}+\cdots+r_1\alpha^3+r_0$$
$$S_4=R(\alpha^4)=r_6\alpha^{24}+r_5\alpha^{20}+\cdots+r_1\alpha^4+r_0$$

where α is the generator of GF(2^4).

Supposing the error location is i_1, i_2, the corresponding error polynomial is:

$$e(x)=e_{i_1}x^{i_1}+e_{i_2}x^{i_2}$$

Define $X_i=\alpha^{i}$ as the error locator number.

$$S_1=e_{i_1}X_1+e_{i_2}X_2$$
$$S_2=e_{i_1}X_1^2+e_{i_2}X_2^2$$
$$S_3=e_{i_1}X_1^3+e_{i_2}X_2^3$$
$$S_4=e_{i_1}X_1^4+e_{i_2}X_2^4$$

2. Find out the error location.

The polynomial to define the error location is:

$$\sigma(x)=\prod_{i=1}^{2}\left(1-X_l x^i\right)=\sigma_2 x^2+\sigma_1 x+1$$

Get solution of the formula:

$$S_1\sigma_2+S_2\sigma_1+S_3=0$$
$$S_2\sigma_2+S_3\sigma_1+S_4=0$$

Generateσ_1 ,σ_2 , put them into the polynomial and detect error locationX_l by Chine search algorithm.

3. calculate the error value e_{i_l} , and correct the error.

Annex F
(informative)

Autodiscrimination compatibility

Han Xin Code can be read by suitably programmed decoders which have been designed to autodiscriminate it from other symbologies. A properly programmed Han Xin Code reader will not decode any ISO/IEC symbologies as a valid Han Xin Code symbol, however, representations of short linear symbols may be found in any matrix symbol including Han Xin Code.

The decoder's valid set of symbologies should be limited to those needed by a given application to maximize reading security.

Annex G
(informative)

Error correction decoding algorithm

Take an example of (n,k,2t) Reed-Solomon code under GF(2^8) is used for error correction.

Provided that the codeword after releasing data masking from the symbol is

$$R=(r_{n-1},r_{n-2},\ldots, r_1,r_0)$$

that is,

$$R=r_{n-1}x^{n-1}+r_{n-2}x^{n-2}+\ldots+r_1x+r_0$$

where r_i (i=0,...,n-1) is the element of the Galois field GF(2^8).

1. Calculate the syndrome.

$$S1=R(\alpha)=r_{n-1}\alpha^{n-1}+r_{n-2}\alpha^{n-2}+\ldots+r_1\alpha+r_0$$

$$S2=R(\alpha 2)=r_{n-1}\alpha^{2(n-1)}+r_{n-2}\alpha^{2(n-2)}+\ldots+r_1\alpha^2+r_0$$

...

$$S1=R(\alpha^{2t})=r_{n-1}\alpha^{2t(n-1)}+r_{n-2}\alpha^{2t(n-2)}+\ldots+r_1\alpha^{2t}+r_0$$

where is the generator of GF(2^8)

Supposing the error position is $i_1, i_2, , i_v$, the corresponding polynomial is:

$e(x)=e_{i_1}x^{i1}+e_{i_2}x^{i2}+\ldots+e_{i_v}x^{iv}$ Define $X_l=\alpha^{i_l}$ as the number of error positions, so:

$$S_1=e_{i_1}X_1+e_{i_2}X_2+\ldots+e_{i_v}X_v$$

$$S_2=e_{i_1}X_1^2+e_{i_2}X_2^2+\ldots+e_{i_v}X_v^2$$

...

$$S_{2t}=e_{i_1}X_1^{2t}+e_{i_2}X_2^{2t}+\ldots+e_{i_v}X_v^{2t}$$

2. Find the Error Position.

Construct the error locator polynomial as follows:

$$\tilde{A}(x)=\prod_{i=1}^{v}\left(1-X_lx^i\right)=\sigma_vx^v+\sigma_{v-1}x^{v-1}+\ldots+\sigma_1x+1$$

Get solution of the formula:

$$S_1\sigma_t+S_2\sigma_{t-1}+\ldots+S_t\sigma_1+S_{t+1}=0$$

$$S_2\sigma_t+S_3\sigma_{t-1}+\ldots+S_{t+1}\sigma_1+S_{t+2}=0$$

...

$$S_t\sigma_t+S_{t+1}\sigma_{t-1}+\ldots+S_{2t-1}\sigma_1+S_{2t}=0$$

So σ_i (i=1,...,t), put it into the locator polynomial, get the error position X_l using Chien search algorithm.

3. Calculate the corresponding error value e^{i_l} , and correct it.

Annex H
(informative)

User guidance for Han Xin Code printing and scanning

H.1 General principles

The application of Han Xin Code is a comprehensive system in which symbol encoding, decoding and usage of relative equipment, such as label printer, printer, reader and labels, should be considered as a whole. Problems in any section or between any two sections may affect the whole system's running.

Except for the critical standard, there are some other factors may affects the system. The following guiding principle lists factors should be considered when adopting barcode or matrix code.

1. Choose appropriate printing density, whose allowable deviation could be covered by label printer or printing technology. Ensure the module dimension is integral multiple of printer pinhead's matrix dimension; ensure the printing increment/reduction could be adjusted. The adjustment is realized by changing the same integral quantity of pixels of each or every adjacent dark modules. The bitmap size of each dark (or light) module changes, the space between every two module centers keep the same.

2. A reader's resolution and read range should be selected based on the symbol's density and the printing technology used.

3. Ensure that sensitivity of the sensor is appropriate to the optical features of the printed bar code symbols.

4. Check and ensure the final labels and barcodes on package are verified in accordance with Clause 8 with aperture and wavelength appropriate to the application. It would affect the symbol's readability if the symbol is covered, penetrated, bended or with irregular surface.

Symbol surface's specular reflection properties should not be neglected. The scanning system shall consider how much the diffuse reflection between dark feature and light feature should be changed. In some scanning angles, the specular reflection is much bigger than the expected diffuse reflectivity, thus changing the scanning features. If changing material or material's surface is possible, it could reduce specular reflection by choosing a rough or unglazed surface. Or, do ensure the illumination for symbol reading could make the symbol reach its best resolution.

H.2 Selection of Error Correction Level

User should choose the proper error correction level to meet the symbol application need. As shown in Table B1, the error correction capacity changes increasingly from L1 to L4; while the size of symbol denoting length-fixed data will increase. For example, symbol of version 44 and error correction level 4 has 509 data codewords. If a lower error correction level is acceptable, the same data could be denoted by a symbol of version 28 and error correction level 1. (The accurate data capacity is 523 codewords.)

The following factors should be considered when choosing symbol's error correction level:

— The expected symbol quality level: the lower the expected symbol quality level is, the higher the error correction level should be;

— The readability of the first time scanning;

— Whether have an opportunity to scan once more if fail;

— Not use higher error correction level resulted in the space limitation for symbol printing.

Error correction level 1 is suitable for symbol with high quality or symbol denoting length-fixed data with smallest size. Level 2 is the standard level and a compromise between small size and reliability. Level 3 is a level with high reliability, suitable for important symbol or symbol with poor quality. Level 4 have the realizably highest reliability.

Annex I
(normative)

Print quality of Han Xin Code — Symbology-specific aspects

I.1 General

This Annex describes the grading approaches for Fixed Pattern damage and Structural Information damage when assessing the printing quality of Han Xin Code symbols according to ISO/IEC 15415.

I.2 Fixed Pattern damage

I.2.1 Testing region

The testing region of Fixed Pattern damage is as shown in Figure I.1.

1. The squares with size of 11×11 modules in the symbol's four corners, each of which includes:
    a. The Position Detection Patterns with size of 7×7 modules.
    b. The Position Detection Pattern separators within the symbol, with width of one module and surrounding the Position Detection Patterns.
    c. The quiet zone outside of the symbol, with width of 3 modules and surrounding the Position Detection Patterns.
2. The Alignment Pattern and Assistant Alignment Pattern in the symbols of version 4 and above.

The testing region mentioned shall be divided into 5 regions:

— Each Position Detection Pattern and its Position Detection Pattern separators and quiet zone form a testing region, thus there are four regions name A_1, A_2, A_3 and A_4.

— All of the Alignment Pattern and Assistant Alignment Patterns in symbols of version 4 and above form region B as the fifth region.

— All of the Assistant Alignment Pattern in symbols of version 4 form region C as the sixth region.

Take Han Xin Code symbol of version 24 with size of 69×69 modules as an example. There are 484 modules in region A, 550 modules in region B, 36 modules in region C.

A_1, A_2, A_3 and A_4 are four testing regions with Position Detection Pattern. B is a testing region comprised by the Alignment Pattern. C is a testing region with and Assistant Alignment Patterns.

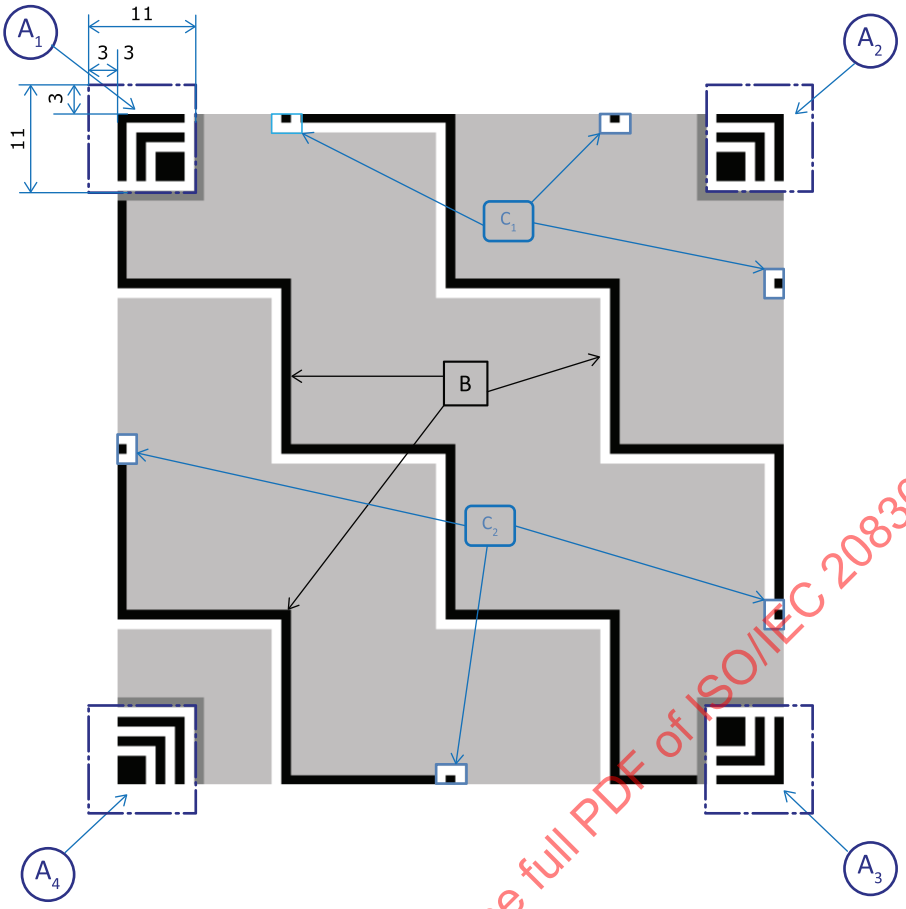


Figure I.1 — The testing region of Fixed Pattern region for Han Xin Code symbols

I.2.2 Grading of Fixed Pattern damage

The grading of Fixed Pattern Damage of each testing region should be based on each module's modulation, including steps as follows:

1. Use approaches described in ISO/IEC 15415 to calculate the grade of each module's modulation. Since the dark/light feature of each module is known already, if the dark module's reflectivity is higher than the global threshold or the light one's reflectivity is lower than the global threshold, the module's modulation grade is 0.

2. Suppose that a modulation grade is determined, assume all modules with lower grades are errors. According to Table I.1 of modulation grade's threshold, ascertain an assumed damage grade. Take the lower value between the modulation grade and the damage grade. The assumed damage grade shall be decided through the approach below:

    a. Count the error modules in A_1, A_2, A_3 and A_4.

    b. In Alignment Pattern's dark line in region B, if there are three assumed error modules consecutive, or less than four modules between error area modules, the module modulation grade should be 0 at step 1.

    c. Count the error modules in Alignment Pattern and Assistant Alignment Patterns in region B, compare with the total quantity of Alignment Pattern and Assistant Alignment Pattern with error modules in the symbol, and record the proportion.

d. According to Table I.1 of region grade's threshold, ascertain a Fixed Pattern damage grade of each testing region.

3. The Fixed Pattern grade for testing regions is the highest among all of the modulation grades. Take the lowest among the five regions' grade as the symbol's Fixed Pattern damage grade.

Table I.1 — Threshold of Fixed Pattern Region grades of Han Xin Code

A_1、A_2、A_3 and A_4	Region B	Region C	
Number of the Error Modules	Proportion of error modules (MD)	Proportion of Alignment Pattern and Assistant Alignment Pattern with error modules	Grade
0	MD=0%	0%	4
1	0%<MD≤9%	≤10%	3
2	9%<MD≤13%	≤20%	2
3	13%<MD≤17%	≤30%	1
4	MD>17%	>30%	0

I.3 Grading of Structural Information region

I.3.1 General

The Structural Information region of the Han Xin Code is a combination of modules denoting symbol version, error correction level and masking solution. Regions in the top left and top right corners of the symbol belong to one group; regions in the bottom left and bottom right corners belong to another group. This region information shall be decoded at the beginning of the decoding process or the process cannot go on. Therefore, grading of the Structural Information region is part of the grading of the whole symbol.

I.3.2 Grading of Structural Information in one group

Supposing the Structural Information in one region group could not be decoded, the group's Structural Information grade is 0. If it could be decoded, the grading should follow steps below.

1. Calculate each codeword's modulation grade according to the approaches described in ISO/IEC 15415. Since the dark/light feature of each module is already known, if the dark module's reflectivity is higher than the global threshold or the light one's reflectivity is lower than the global threshold, the module's modulation grade is 0.

2. The cumulative number of codewords achieving each grade, from 4 to 0 or a higher grade, shall be counted and the counts shall be compared with the error correction capacity of the group as follows.

   a) For each grade level, assuming that all codewords not achieving that grade or a higher grade are errors, derive a notional Unused Error Correction grade as described in ISO/IEC 15415:2011, 7.8.8. Take the lower of the grade level and the notional UEC grade.

   b) Then the Structural Information grade for the group shall be the highest of the resulting values for all MOD grade levels.

Table I.2 shows an example of the Structural Information grading.

Table I.2 — Grading of Structural Information in one group

MOD codeword grade level (**a**)	No. of codewords at level **a**	Cumulative no. of codewords at level **a** or higher (**b**)	Remaining codewords (treated as errors) (7 - **b**) (**c**)	Notional unused error correction capacity (4 - 2**c**)	Notional UEC (%)	Notional UEC grade (**d**)	Lower of **a** or **d** (**e**)
4	4	4	3	N/A	N/A	0	0
3	2	6	1	2	50%	3	3
2	0	6	1	2	50%	3	2
1	0	6	1	2	50%	3	1
0	1	7	0	4	100%	4	0
Structural Information grade (Highest value of **e**):							3

I.3.3 Grading of Structural Information region

The lower of these two groups' grade is the symbol's Structural Information region grade.

I.4 Scan grade

The single scan grade shall be the lowest value among decode, modulation grade, axial non-uniformity, grid non-uniformity, unused error correction described in ISO/IEC 15415 and Structural Information region and Fixed Pattern damage grades described in this annex.

Annex J
(informative)

Useful process control techniques

J.1 General

This annex describes tools and procedures useful for monitoring and controlling the process of creating scannable Han Xin Code symbols. These techniques do not constitute a print quality check of the produced symbols (the method of Clause 8 and Annex I 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 hence is useful for process control.

J.2 Symbol contrast

Most linear bar code verifiers have a reflectometer mode, a mode for plotting scan reflectance profiles or reporting Symbol Contrast (the verification process shall conform to ISO/IEC 15416) from undecodable scans. In general the symbol contrast readings that can be obtained using a 0,25 mm (0.010 inch) aperture at 660 nm wavelength (either the reported Symbol Contrast value, the peak-to- peak scan profile excursions, or the difference between peak reflectometer readings) are found to correlate well with an image-derived symbol contrast. In particular these readings can be used to check that symbol contrast stays well above the minimum allowed for the intended symbol quality grade and whether it is changing appreciably during a print run.

J.3 Assessing Axial Nonuniformity

For a Han Xin Code symbol, measure the width and height of the symbol. Divide width by height and assess the result for Axial Nonuniformity.

J.4 Visual inspection for symbol distortion and defects

Ongoing visual inspection of the Finder Pattern in sample symbols can monitor an important aspect of the print process.

Matrix code symbols are susceptible to errors caused by local distortions of the matrix grid. Any such distortions may show up visually as either crooked edges on the Finder Pattern or uneven spacing within the Alignment Pattern and Assistant Alignment Pattern and aligned with the inner boundaries of these.

The Finder Pattern and the adjacent Quiet Zone areas should always be solidly dark and light. Failures in the print mechanism which may produce defects in the form of light or dark streaks through the symbol should be visibly evident where they traverse the Finder Pattern or the Quiet Zone. Such systematic failures in the print process should be corrected.

Look for isolated light and dark single modules to ensure they are the same size as other modules.

Annex K
(informative)

Han Xin Code encoding examples

K.1 General

This Annex describes two encoding examples for the Han Xin Code. The first one is the encoding example that does not have user selected version and user selected error correction level, in order to show how the Han Xin Code encoding algorithm shall be followed. The second one illustrates the user selected version (10) and user selected error correction level (L3) to encode the data to the Han Xin Code symbol.

K.2 Example 1

The data to be encoded is:

1234567890 (K.1.0)

Step 1. Data analysis

According to 5.2 and 5.3, the data (K.1.0) shall be encoded in the encoding modes as shown in Table K.1.

Table K.1 — Encoding modes after data analysis

Data	1	2	3	4	5	6	7	8
Byte Value	$(31)_{hex}$	$(32)_{hex}$	$(33)_{hex}$	$(34)_{hex}$	$(35)_{hex}$	$(36)_{hex}$	$(37)_{hex}$	$(38)_{hex}$
Encoding Mode	Numeric mode							
Data	9	0						
Byte Value	$(39)_{hex}$	$(30)_{hex}$						
Encoding Mode	Numeric mode							

Step 2. Data encoding

According to 5.4 and Table K.1**.1**, The data (K.1.0) shall be encoded to the information bit stream (K.1.1):

00010001 11101101 11001000 11000101 01000000 00001111 111101 (K.1.1)

The length of the information bit stream (K.1.1) is 54 bits. This data shall be encoded in the Version 1 Han Xin Code symbol with error correction level L1.

Step 3. Generating the error correction codewords and construct final data bit stream

According to 5.5 and Annex B, the information bit stream (K.1.1) shall be converted to the information codeword stream (K.1.2):

$(11)_{hex}$ $(ED)_{hex}$ $(C8)_{hex}$ $(C5)_{hex}$ $(40)_{hex}$ $(0F)_{hex}$ $(F4)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ (K.1.2)

There are 21 codewords in the information codeword stream (L.1.2).

According to Annex B, there is 1 block in Version 1 L1, and the error correction characteristic is:

(25,21,4)

Utilising the Reed-Solomon error correction algorithm described in 5.5, the error correction codewords shall be generated. According to the 5.7, re-assemble information codewords and error correction codewords to the final data codeword stream (K.1.3):

$(11)_{hex}$ $(ED)_{hex}$ $(C8)_{hex}$ $(C5)_{hex}$ $(40)_{hex}$ $(0F)_{hex}$ $(F4)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ (K.1.3)
$(00)_{hex}$ $(EB)_{hex}$ $(B4)_{hex}$ $(68)_{hex}$ $(1D)_{hex}$

The length of the final data codeword stream (K.1.3) is 25 codewords.

Step 4. Function Pattern placement

According to the 5.8.2, the Symbol of Version 1 is shown in Figure K.1.

Key

dark module

light module

function information region

information encoding region

Figure K.1 — Version 1 Han Xin Code symbol

Step 5. Data placement

The data codeword stream (K.1.3) shall be broken up using the algorithm described in 5.8.3, and hence re-assembled to the data bit stream (L.1.4) with 200 bits:

00010001 00000000 11101101 00000000 11001000 00000000 11000101
00000000 01000000 00000000 00001111 00000000 11110100 00000000
00000000 00000000 00000000 11101011 00000000 10110100 00000000 (L.1.4)
01101000 00000000 00011101 00000000

The data bit stream (L.1.4) shall be placed in the symbol with the order shown in Figure K.2:

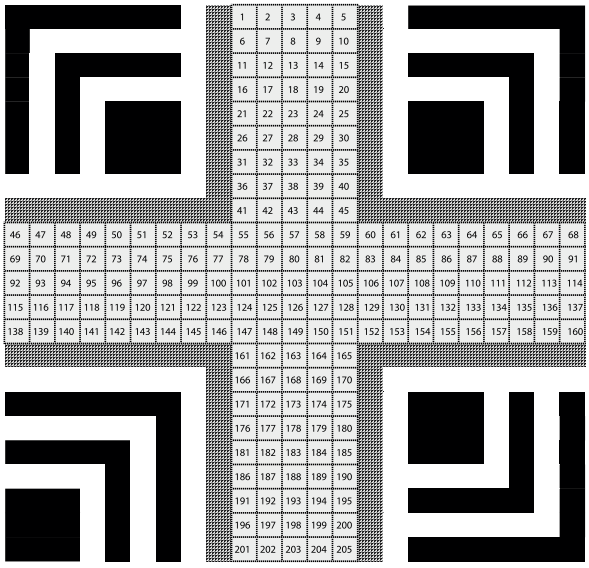


Figure K.2 — Data stream placement in Version 1 Han Xin Code symbol

The result after data placement is shown in Figure K.3.

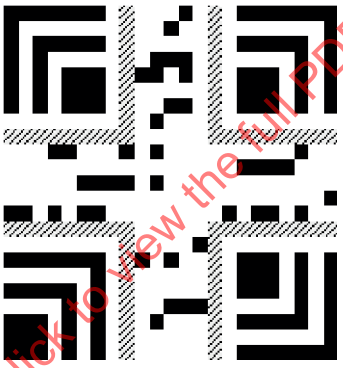


Figure K.3 — Han Xin Code symbol after data placement

Step 6. Masking

Utilising the masking patterns in 5.8.4 to mask Figure K.3. See Figure K.4 to Figure K.7.

Figure K.4 — Masking result for Figure K.3 with masking pattern 00

Figure K.5 — Masking result for Figure K.3 with masking pattern 01

Figure K.6 — Masking result for Figure K.3 with masking pattern 10

Figure K.7 — Masking result for Figure K.3 with masking pattern 11

Calculate the penalty scores of the patterns in accordance with Table K.2.

Table K.2 — Penalty scores of masking patterns

Condition of masking solution	Data mask pattern reference for Han Xin Code	Penalty score
Non-masking	00	3584
(i+j) mod 2=0	01	3120
((i+j)mod 3+(j mod 3)) mod 2=0	10	3240
(i mod j +j mod i + i mod 3+ j mod 3) mod 2=0	11	3410

The masking solution of this Han Xin Code symbol shall be 01.

Step 7. Generation of the function information

The function information encoding is illustrated in Table K.3.

Table K.3 — Encoding of function information

	Encoding for function information of the Han Xin Code symbol
Version	20+1=21, encoded as 0001 0101
Error correction level	00
Masking solution	01

Utilising the algorithm in Annex E, generate the function information (K.1.5):

0001 0101 0001 0101 0011 1100 1011 (K.1.5)

Step 8. Function information placement

Utilising the algorithm in 5.8.5, place the function information in the function information region, and get the final Han Xin Code symbol of this example (see Figure K.8).

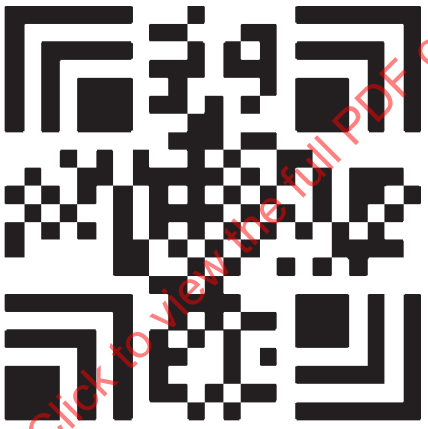


Figure K.8 — Final Han Xin Code symbol

K.3 Example 2

The data to be encoded is:

1234567890ABCDEFGabcdefg,Han Xin Code (K.2.0)

The user chosen version is 10, and the user chosen error correction level is L3.

Step 1. Data analysis

According to 5.2 and 5.3, the data (K.2.0) shall be encoded in the encoding modes in Table K.4.

Table K.4 — Encoding modes after data analysis

Data	1	2	3	4	5	6	7	8
Byte Value	$(31)_{hex}$	$(32)_{hex}$	$(33)_{hex}$	$(34)_{hex}$	$(35)_{hex}$	$(36)_{hex}$	$(37)_{hex}$	$(38)_{hex}$
Encoding Mode	Numeric mode							

Table K.4 *(continued)*

Data	9	0	A	B	C	D	E	F
Byte Value	$(39)_{hex}$	$(30)_{hex}$	$(41)_{hex}$	$(42)_{hex}$	$(43)_{hex}$	$(44)_{hex}$	$(45)_{hex}$	$(46)_{hex}$
Encoding Mode	Numeric mode		Text 1 of Text mode					
Data	G	a	b	c	d	e	f	g
Byte Value	$(47)_{hex}$	$(61)_{hex}$	$(62)_{hex}$	$(63)_{hex}$	$(64)_{hex}$	$(65)_{hex}$	$(66)_{hex}$	$(67)_{hex}$
Encoding Mode	Text 1 of Text mode							
Data	,	H	a	n		X	i	n
Byte Value	$(2C)_{hex}$	$(48)_{hex}$	$(61)_{hex}$	$(6E)_{hex}$	$(20)_{hex}$	$(58)_{hex}$	$(69)_{hex}$	$(6E)_{hex}$
Encoding Mode	Text 2 of Text mode	Text 1 of Text mode			Text 2 of Text mode	Text 1 of Text mode		
Data		C	o	d	e			
Byte Value	$(20)_{hex}$	$(43)_{hex}$	$(6F)_{hex}$	$(64)_{hex}$	$(65)_{hex}$			
Encoding Mode	Text 2 of Text mode	Text 1 of Text mode						

Step 2. Data encoding

According to 5.4 and Table K.4, The data (K.2.0) shall be encoded to the information bit stream (K.2.1):

00010001 11101101 11001000 11000101 01000000 00001111 11110100
10001010 00101100 11000011 01001110 00110101 00001001 00100101
10011010 01111010 00101001 10101011 11101010 00111110 01000110 (K.2.1)
01001100 01111110 01110011 11101000 01101100 11000111 11100111
00111110 00110011 00101001 11101000 111111

The length of the information bit stream (K.2.1) is 262 bits.

Step 3. Generating the error correction codewords and construct final data bit stream

According to 5.5, the information bit stream (K.2.1) shall be converted to the information codeword stream (K.2.2):

$(11)_{hex}$ $(ED)_{hex}$ $(C8)_{hex}$ $(C5)_{hex}$ $(40)_{hex}$ $(0F)_{hex}$ $(F4)_{hex}$ $(8A)_{hex}$ $(2C)_{hex}$ $(C3)_{hex}$
$(4E)_{hex}$ $(3D)_{hex}$ $(09)_{hex}$ $(25)_{hex}$ $(9A)_{hex}$ $(7A)_{hex}$ $(29)_{hex}$ $(AB)_{hex}$ $(EA)_{hex}$ $(3E)_{hex}$
$(46)_{hex}$ $(4C)_{hex}$ $(7E)_{hex}$ $(73)_{hex}$ $(E8)_{hex}$ $(6C)_{hex}$ $(C7)_{hex}$ $(E7)_{hex}$ $(3E)_{hex}$ $(33)_{hex}$
$(29)_{hex}$ $(E8)_{hex}$ $(FC)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ (K.2.2)
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$

There are 83 codewords in the information codeword stream (K.2.2).

According to Annex B, there are 3 blocks in Version 10 with error correction level K3, and the error correction characteristics of each blocks are:

(51,27,24), (51,27,24), (53,29,24)

The information codewords shall be divided into 3 blocks, which contain 27, 27 and 29 codewords. Utilising the Reed-Solomon error correction algorithm described in 5.5 in each block, the error

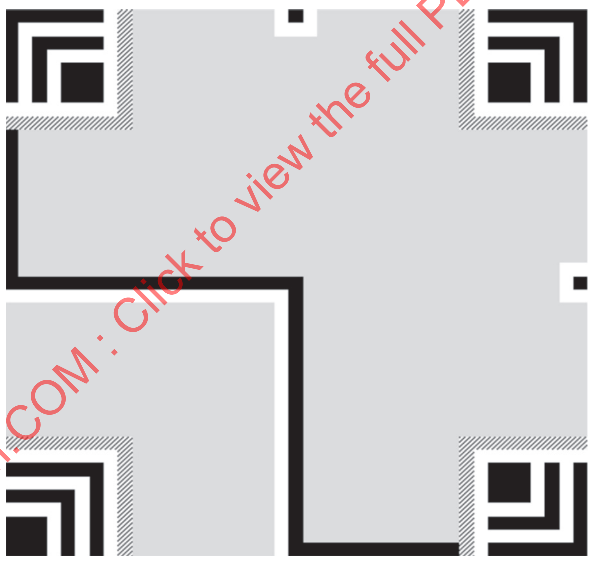
correction codewords shall be generated. According to the 5.7, re-assemble information codewords and error correction codewords to the final data codeword stream (K.2.3):

$$(11)_{hex}\ (ED)_{hex}\ (C8)_{hex}\ (C5)_{hex}\ (40)_{hex}\ (0F)_{hex}\ (F4)_{hex}\ (8A)_{hex}\ (2C)_{hex}\ (C3)_{hex}\ (4E)_{hex}\ (3D)_{hex}\ (09)_{hex}\ (25)_{hex}\ (9A)_{hex}\ (7A)_{hex}\ (29)_{hex}\ (AB)_{hex}\ (EA)_{hex}\ (3E)_{hex}\ (46)_{hex}\ (4C)_{hex}\ (7E)_{hex}\ (73)_{hex}\ (E8)_{hex}\ (6C)_{hex}\ (C7)_{hex}\ (08)_{hex}\ (57)_{hex}\ (0C)_{hex}\ (E0)_{hex}\ (7A)_{hex}\ (A5)_{hex}\ (DD)_{hex}\ (A2)_{hex}\ (99)_{hex}\ (CF)_{hex}\ (A4)_{hex}\ (82)_{hex}\ (AD)_{hex}\ (11)_{hex}\ (B0)_{hex}\ (84)_{hex}\ (74)_{hex}\ (5D)_{hex}\ (9A)_{hex}\ (99)_{hex}\ (0B)_{hex}\ (CD)_{hex}\ (49)_{hex}\ (77)_{hex}\ (E7)_{hex}\ (3E)_{hex}\ (33)_{hex}\ (29)_{hex}\ (E8)_{hex}\ (FC)_{hex}\ (00)_{hex}\ (A2)_{hex}\ (A7)_{hex}\ (68)_{hex}\ (8A)_{hex}\ (5F)_{hex}\ (E6)_{hex}\ (AA)_{hex}\ (11)_{hex}\ (A6)_{hex}\ (69)_{hex}\ (4A)_{hex}\ (CF)_{hex}\ (CF)_{hex}\ (20)_{hex}\ (5D)_{hex}\ (00)_{hex}\ (1B)_{hex}\ (79)_{hex}\ (A1)_{hex}\ (FE)_{hex}\ (B7)_{hex}\ (94)_{hex}\ (03)_{hex}\ (9B)_{hex}\ (00)_{hex} \tag{K.2.3}$$

The length of the final data codeword stream (K.2.3) is 155 codewords.

Step 4. Function Pattern placement

According to 5.8.2, Function Pattern Placement for the Symbol of Version 10 is shown in Figure K.9.



Key

dark module

light module

function information region

information encoding region

Figure K.9 — Function Pattern placement

Step 5.Data placement

The data codeword stream (K.2.3) shall be broken up using the algorithm described in 5.8.3, and hence re-assembled to the data bit stream (K.2.4) with 1240 bits:

00010001 00100101 11000111 10101101 00111110 00000000 10100010
00100000 00000000 00000000 00000000 00000000 11101101 10011010
00001000 00010001 00110011 00000000 10100111 01011101 00000000
00000000 00000000 00000000 11001000 01111010 01010111 10110000
00101001 00000000 01101000 00000000 00000000 00000000 00000000
00000000 11000101 00101001 00001100 10000100 11101000 00000000
10001010 00011011 00000000 00000000 00000000 00000000 01000000
10101011 11100000 01110100 11111100 00000000 01011111 01111001
00000000 00000000 00000000 00000000 00001111 11101010 01111010
01011101 00000000 00000000 11100110 10100001 00000000 00000000
00000000 00000000 11110100 00111110 10100101 10011010 00000000
00000000 10101010 11111110 00000000 00000000 00000000 00000000 (K.2.4)
10001010 01000110 11011101 10011001 00000000 00000000 00010001
10110111 00000000 00000000 00000000 00000000 00101100 01001100
10100010 00001011 00000000 00000000 10100110 10010100 00000000
00000000 00000000 00000000 11000011 01111110 10011001 11001101
00000000 00000000 01101001 00000011 00000000 00000000 00000000
00000000 01001110 01110011 11001111 01001001 00000000 00000000
01001010 10011011 00000000 00000000 00000000 00000000 00111101
11101000 10100100 01110111 00000000 00000000 11001111 00000000
00000000 00000000 00000000 00000000 00001001 01101100 10000010
11100111 00000000 00000000 11001111 00000000 00000000 00000000
00000000

The data bit stream (K.2.4) shall be placed in the symbol with the order shown in Figure K.10:

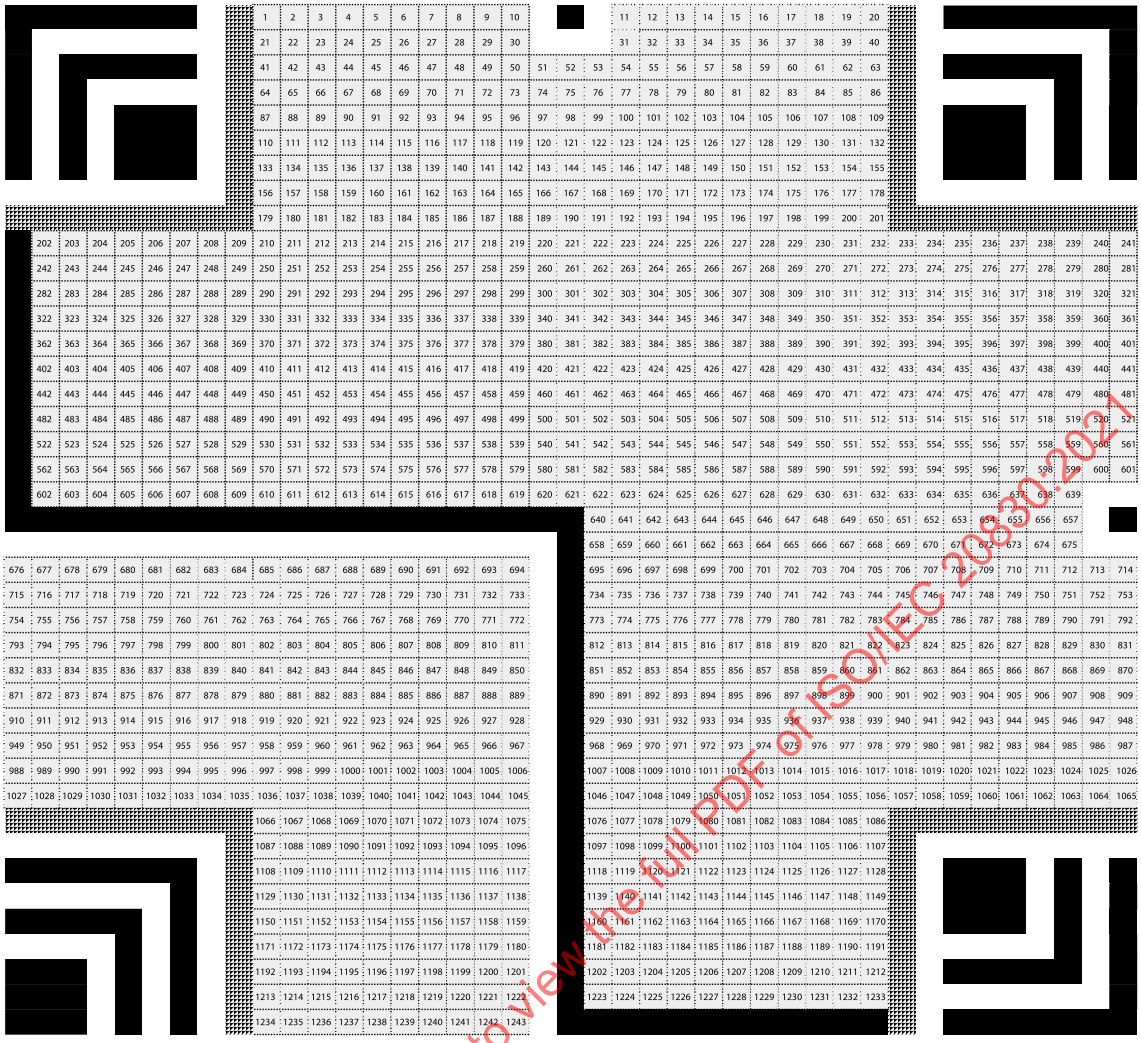


Figure K.10 — Placement of the data bit stream

The result after data placement is shown as Figure K.11.

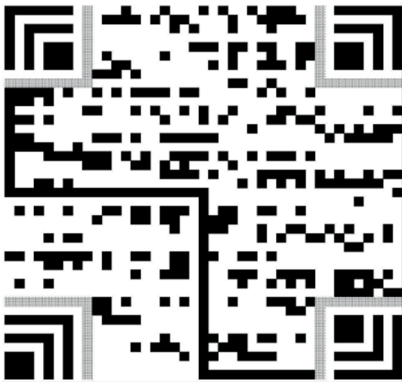


Figure K.11 — Han Xin Code symbol Version 10 after data placement (K.2)

Step 6. Masking

Utilising the masking patterns in 5.8.4 to mask Figure K.11. See Figure K.12 to Figure K.15.

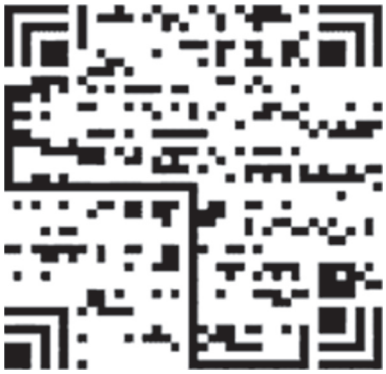


Figure K.12 — Masking result for Figure K.11 with masking pattern 00

Figure K.13 — Masking result for Figure K.11 with masking pattern 01

Figure K.14 — Masking result for Figure K.11 with masking pattern 10

Figure K.15 — Masking result for Figure K.11 with masking pattern 11

Calculate the penalty scores of the patterns in accordance with Table K.5.

Table K.5 — Penalty scores of masking patterns

Condition of masking solution	Data mask pattern reference for Han Xin Code	Penalty score
Non-masking	00	9868
(i+j) mod 2=0	01	7408
((i+j)mod 3+(j mod 3)) mod 2=0	10	6888
(i mod j +j mod i + i mod 3+ j mod 3) mod 2=0	11	8652

The masking solution of this Han Xin Code symbol shall be 10.

Step 7. Generation of the function information

The function information encoding is illustrated in Table K.6.

Table K.6 — Encoding of function information

	Encoding for function information of the Han Xin Code symbol
Version	20+10=30, encoded as 0001 1110
Error correction level	10
Masking solution	10

Utilising the algorithm in Annex E, generate the function information (K.2.5):

0001 1110 1010 0110 0111 1010 1110 (K.2.5)

Step 8. Function information placement

Utilising the algorithm in 5.8.5, place the function information in the function information region, and get the final Han Xin Code symbol of this example (see Figure K.16).

Figure K.16 — Final Han Xin Code symbol (K.2)

K.4 Example 3

The data to be encoded is:

Summer Palace Ticket for 6 June 2015 13:00;2015年6月6日夜01時00分PM (K.3.0)
頤和園のチケット;2015□6□6□13□□□□□□□□□□.2015年6月6号下午13:00的颐和园门票;

The user chosen version is 17, and the user chosen error correction level is L2.

Step 1. Data analysis

According to 5.2 and 5.3, the data (K.3.0) shall be encoded in the encoding modes in Table K.7.

Table K.7 — Encoding modes after data analysis

Data	S	u	m	m	e	r
Byte value	$(53)_{hex}$	$(75)_{hex}$	$(6d)_{hex}$	$(6d)_{hex}$	$(65)_{hex}$	$(72)_{hex}$
Encoding mode	Text 1 of Text mode					
Data		P	a	l	a	c
Byte value	$(20)_{hex}$	$(50)_{hex}$	$(61)_{hex}$	$(6c)_{hex}$	$(61)_{hex}$	$(63)_{hex}$
Encoding mode	Text 2 of Text mode	Text 1 of Text mode				
Data	e		T	i	c	k
Byte value	$(65)_{hex}$	$(20)_{hex}$	$(54)_{hex}$	$(69)_{hex}$	$(63)_{hex}$	$(6b)_{hex}$
Encoding mode	Text 1 of Text mode	Text 2 of Text mode	Text 1 of Text mode			
Data	e	t		f	o	r
Byte value	$(65)_{hex}$	$(74)_{hex}$	$(20)_{hex}$	$(66)_{hex}$	$(6f)_{hex}$	$(72)_{hex}$
Encoding mode	Text 1 of Text mode		Text 2 of Text mode	Text 1 of Text mode		
Data		6		J	u	n
Byte value	$(20)_{hex}$	$(36)_{hex}$	$(20)_{hex}$	$(4a)_{hex}$	$(75)_{hex}$	$(6e)_{hex}$
Encoding mode	Text 2 of Text mode	Text 1 of Text mode	Text 2 of Text mode	Text 1 of Text mode		
Data	e		2	0	1	5
Byte value	$(65)_{hex}$	$(20)_{hex}$	$(32)_{hex}$	$(30)_{hex}$	$(31)_{hex}$	$(35)_{hex}$

Table K.7 *(continued)*

Encoding mode	Text 1 of Text mode	Text 2 of Text mode	Text 1 of Text mode									
Data		1	3	:	0	0						
Byte value	$(20)_{hex}$	$(31)_{hex}$	$(33)_{hex}$	$(3a)_{hex}$	$(30)_{hex}$	$(30)_{hex}$						
Encoding mode	Text 2 of Text mode	Text 1 of Text mode			Text 2 of Text mode	Text 1 of Text mode						
Data	;	2	0	1	5	年						
Byte value	$(3b)_{hex}$	$(32)_{hex}$	$(30)_{hex}$	$(31)_{hex}$	$(35)_{hex}$	$(c4)_{hex}$	$(ea)_{hex}$					
Encoding mode	Text 2 of Text mode	Text 1 of Text mode				Common Chinese Characters in Region One mode						
Data	6	月		6	日							
Byte value	$(36)_{hex}$	$(d4)_{hex}$	$(c2)_{hex}$	$(36)_{hex}$	$(c8)_{hex}$	$(d5)_{hex}$						
Encoding mode	Text 1 of Text mode	Common Chinese Characters in Region One mode		Text 1 of Text mode	Common Chinese Characters in Region One mode							
Data	夜		0	1	時							
Byte value	$(d2)_{hex}$	$(b9)_{hex}$	$(30)_{hex}$	$(31)_{hex}$	$(95)_{hex}$	$(72)_{hex}$						
Encoding mode	Common Chinese Characters in Region One mode		Text 1 of Text mode		GB18030 2-byte Region mode							
Data	0	0	分		P	M						
Byte value	$(30)_{hex}$	$(30)_{hex}$	$(b7)_{hex}$	$(d6)_{hex}$	$(50)_{hex}$	$(4d)_{hex}$						
Encoding mode	Text 1 of Text mode		Common Chinese Characters in Region One mode		Text 1 of Text mode							
Data	頤		和		園							
Byte value	$(ee)_{hex}$	$(55)_{hex}$	$(ba)_{hex}$	$(cd)_{hex}$	$(88)_{hex}$	$(40)_{hex}$						
Encoding mode	GB18030 2-byte Region mode											
Data	の		チ		ケ							
Byte value	$(a4)_{hex}$	$(ce)_{hex}$	$(a5)_{hex}$	$(c1)_{hex}$	$(a5)_{hex}$	$(b1)_{hex}$						
Encoding mode	GB18030 2-byte Region mode											
Data	ッ		ト		;	2	0					
Byte value	$(a5)_{hex}$	$(c3)_{hex}$	$(a5)_{hex}$	$(c8)_{hex}$	$(3b)_{hex}$	$(32)_{hex}$	$(30)_{hex}$					
Encoding mode	GB18030 2-byte Region mode				Text 2 of Text mode	Text 1 of Text mode						
Data	1	5	년									
Byte value	$(31)_{hex}$	$(35)_{hex}$	$(82)_{hex}$	$(38)_{hex}$	$(d8)_{hex}$	$(33)_{hex}$						
Encoding mode	Text 1 of Text mode		GB18030 4-byte Region mode									
Data	6	월				6						
Byte value	$(36)_{hex}$	$(83)_{hex}$	$(33)_{hex}$	$(8a)_{hex}$	$(33)_{hex}$	$(36)_{hex}$						
Encoding mode	Text 1 of Text mode	GB18030 4-byte Region mode				Text1 of Text mode						
Data	일				1	3						
Byte value	$(83)_{hex}$	$(33)_{hex}$	$(9b)_{hex}$	$(31)_{hex}$	$(31)_{hex}$	$(33)_{hex}$						
Encoding mode	GB18030 4-byte Region mode				Text1 of Text mode							

Table K.7 *(continued)*

Data	시				오			
Byte value	$(83)_{hex}$	$(32)_{hex}$	$(a2)_{hex}$	$(37)_{hex}$	$(83)_{hex}$	$(32)_{hex}$	$(f6)_{hex}$	$(37)_{hex}$
Encoding mode	GB18030 4-byte Region mode							
Data	후				여			
Byte value	$(83)_{hex}$	$(36)_{hex}$	$(a8)_{hex}$	$(33)_{hex}$	$(83)_{hex}$	$(32)_{hex}$	$(f1)_{hex}$	$(31)_{hex}$
Encoding mode	GB18030 4-byte Region mode							
Data	름				궁			
Byte value	$(83)_{hex}$	$(30)_{hex}$	$(af)_{hex}$	$(35)_{hex}$	$(82)_{hex}$	$(37)_{hex}$	$(f6)_{hex}$	$(30)_{hex}$
Encoding mode	GB18030 4-byte Region mode							
Data	전				티			
Byte value	$(83)_{hex}$	$(33)_{hex}$	$(a8)_{hex}$	$(37)_{hex}$	$(83)_{hex}$	$(35)_{hex}$	$(c4)_{hex}$	$(33)_{hex}$
Encoding mode	GB18030 4-byte Region mode							

Data	켓				.	2	0
Byte value	$(83)_{hex}$	$(34)_{hex}$	$(df)_{hex}$	$(34)_{hex}$	$(2e)_{hex}$	$(32)_{hex}$	$(30)_{hex}$
Encoding mode	GB18030 4-byte Region mode				Text 2 of Text mode	Text 1 of Text mode	

Data	1	5	年		6	月	
Byte value	$(31)_{hex}$	$(35)_{hex}$	$(c4)_{hex}$	$(ea)_{hex}$	$(36)_{hex}$	$(d4)_{hex}$	$(c2)_{hex}$
Encoding mode	Text 1 of Text mode		Common Chinese Characters in Region One mode		Text 1 of Text mode	Common Chinese Characters in Region One mode	

Data	6	号		下		午	
Byte value	$(36)_{hex}$	$(ba)_{hex}$	$(c5)_{hex}$	$(cf)_{hex}$	$(c2)_{hex}$	$(ce)_{hex}$	$(e7)_{hex}$
Encoding mode	Text 1 of Text mode	Common Chinese Characters in Region One mode					

Data	1	3	:	0	0
Byte value	$(31)_{hex}$	$(33)_{hex}$	$(3a)_{hex}$	$(30)_{hex}$	$(30)_{hex}$
Encoding mode	Text 1 of Text mode		Text 2 of Text mode	Text 1 of Text mode	

Data	的		颐		和	
Byte value	$(b5)_{hex}$	$(c4)_{hex}$	$(d2)_{hex}$	$(c3)_{hex}$	$(ba)_{hex}$	$(cd)_{hex}$
Encoding mode	Common Chinese Characters in Region One mode					

Data	园		门		票		;
Byte value	$(d4)_{hex}$	$(b0)_{hex}$	$(c3)_{hex}$	$(c5)_{hex}$	$(c6)_{hex}$	$(b1)_{hex}$	$(3b)_{hex}$
Encoding mode	Common Chinese Characters in Region One mode						Text 2 of Text mode

Step 2. Data encoding

According to 5.4 and Table K.7, The data (K.3.0) shall be encoded to the information bit stream (K.3.1):

00100111 00111000 11000011 00001010 00110101 11111001 11001111
10011001 10010010 11111001 00100110 10100011 11100111 00111110
01110110 11001001 10101110 10100011 01111111 10011100 11111010
10011100 10110101 11111001 11001111 10000110 11111001 11001111
10010011 11100011 00011010 00111110 01110011 11100000 10000000
00000100 01011111 10011100 11111000 00010000 11111110 10110011
11100000 00000000 11111010 11011111 10000010 00000000 00010001
01111111 01000111 10100001 11111111 11110010 00011011 11110100
11010101 10011111 11111111 00100001 10111111 01001001 00000100
11001001 01001111 11111111 00100000 00000001 11111101 10000111
10000101 01111111 11111111 00100000 00000000 11111101 00001011
00011111 11111111 11001001 10010101 10111111 01101010 00011111
01101010 10110110 10000010 10011001 00011010 10000111 00110110
01110000 01101100 10100000 11011001 11010001 10110011 11111111
11111111 11100101 11110101 10111111 00000100 00000000 00100010
11111110 11100000 01011100 00000001 00100001 10111111 01110000
00111000 11001000 10010000 11011111 10111000 00011100 10001110
01001000 00010000 11111111 01110000 00110110 11001100 10111000
00011100 00111000 01011100 00010000 00110000 00101110 00000111
00001010 10010111 00000011 00100010 00001011 10000001 01100000
11111001 11000000 11100101 10000010 11100000 01111101 10101101
01110000 00111100 11101000 00010111 11010101 01111100 00010000
00000000 10001011 11111010 00111101 00001111 11111111 10000000
11011111 10100110 10101100 11111111 11111001 00001101 11111010
00011110 10000101 11000001 11011010 01010111 11111111 10010000
00100001 11111101 01100111 11000000 00000001 11111010 00001111
11001110 01001111 00011110 11000110 10100011 10111000 11110100
00010010 01111111 11111001 01111101 01101111 11100000

(K.3.1)

The length of the information bit stream (K.3.1) is 1560 bits.

Step 3. Generating the error correction codewords and construct final data bit stream

According to 5.5, the information bit stream (K.3.1) shall be converted to the information codeword stream (K.3.2):

$(27)_{hex}$ $(38)_{hex}$ $(C3)_{hex}$ $(0A)_{hex}$ $(35)_{hex}$ $(F9)_{hex}$ $(CF)_{hex}$ $(99)_{hex}$ $(92)_{hex}$ $(F9)_{hex}$
$(26)_{hex}$ $(A3)_{hex}$ $(E7)_{hex}$ $(3E)_{hex}$ $(76)_{hex}$ $(C9)_{hex}$ $(AE)_{hex}$ $(A3)_{hex}$ $(7F)_{hex}$ $(9C)_{hex}$
$(FA)_{hex}$ $(9C)_{hex}$ $(B5)_{hex}$ $(F9)_{hex}$ $(CF)_{hex}$ $(86)_{hex}$ $(F9)_{hex}$ $(CF)_{hex}$ $(93)_{hex}$ $(E3)_{hex}$
$(1A)_{hex}$ $(3E)_{hex}$ $(73)_{hex}$ $(E0)_{hex}$ $(80)_{hex}$ $(04)_{hex}$ $(5F)_{hex}$ $(9C)_{hex}$ $(F8)_{hex}$ $(10)_{hex}$
$(FE)_{hex}$ $(B3)_{hex}$ $(E0)_{hex}$ $(00)_{hex}$ $(FA)_{hex}$ $(DF)_{hex}$ $(82)_{hex}$ $(00)_{hex}$ $(11)_{hex}$ $(7F)_{hex}$
$(47)_{hex}$ $(A1)_{hex}$ $(FF)_{hex}$ $(F2)_{hex}$ $(1B)_{hex}$ $(F4)_{hex}$ $(D5)_{hex}$ $(9F)_{hex}$ $(FF)_{hex}$ $(21)_{hex}$
$(BF)_{hex}$ $(49)_{hex}$ $(04)_{hex}$ $(C9)_{hex}$ $(4F)_{hex}$ $(FF)_{hex}$ $(20)_{hex}$ $(01)_{hex}$ $(FD)_{hex}$ $(87)_{hex}$
$(85)_{hex}$ $(7F)_{hex}$ $(FF)_{hex}$ $(20)_{hex}$ $(00)_{hex}$ $(FD)_{hex}$ $(0B)_{hex}$ $(1F)_{hex}$ $(FF)_{hex}$ $(C9)_{hex}$
$(95)_{hex}$ $(BF)_{hex}$ $(6A)_{hex}$ $(1F)_{hex}$ $(6A)_{hex}$ $(B6)_{hex}$ $(82)_{hex}$ $(99)_{hex}$ $(1A)_{hex}$ $(87)_{hex}$
$(36)_{hex}$ $(70)_{hex}$ $(6C)_{hex}$ $(A0)_{hex}$ $(D9)_{hex}$ $(D1)_{hex}$ $(B3)_{hex}$ $(FF)_{hex}$ $(FF)_{hex}$ $(E5)_{hex}$
$(F5)_{hex}$ $(BF)_{hex}$ $(04)_{hex}$ $(00)_{hex}$ $(22)_{hex}$ $(FE)_{hex}$ $(E0)_{hex}$ $(5C)_{hex}$ $(01)_{hex}$ $(21)_{hex}$
$(BF)_{hex}$ $(70)_{hex}$ $(38)_{hex}$ $(C8)_{hex}$ $(90)_{hex}$ $(DF)_{hex}$ $(B8)_{hex}$ $(1C)_{hex}$ $(8E)_{hex}$ $(48)_{hex}$
$(10)_{hex}$ $(FF)_{hex}$ $(70)_{hex}$ $(36)_{hex}$ $(CC)_{hex}$ $(B8)_{hex}$ $(1C)_{hex}$ $(38)_{hex}$ $(5C)_{hex}$ $(10)_{hex}$ (K.3.2)
$(30)_{hex}$ $(2e)_{hex}$ $(07)_{hex}$ $(0A)_{hex}$ $(97)_{hex}$ $(03)_{hex}$ $(22)_{hex}$ $(0B)_{hex}$ $(81)_{hex}$ $(60)_{hex}$
$(F9)_{hex}$ $(C0)_{hex}$ $(E5)_{hex}$ $(82)_{hex}$ $(E0)_{hex}$ $(7D)_{hex}$ $(AD)_{hex}$ $(70)_{hex}$ $(3C)_{hex}$ $(E8)_{hex}$
$(17)_{hex}$ $(D5)_{hex}$ $(7C)_{hex}$ $(10)_{hex}$ $(00)_{hex}$ $(8B)_{hex}$ $(FA)_{hex}$ $(3D)_{hex}$ $(0F)_{hex}$ $(FF)_{hex}$
$(90)_{hex}$ $(DF)_{hex}$ $(A6)_{hex}$ $(AC)_{hex}$ $(FF)_{hex}$ $(F9)_{hex}$ $(0D)_{hex}$ $(FA)_{hex}$ $(1E)_{hex}$ $(85)_{hex}$
$(C1)_{hex}$ $(DA)_{hex}$ $(57)_{hex}$ $(FF)_{hex}$ $(90)_{hex}$ $(21)_{hex}$ $(FD)_{hex}$ $(67)_{hex}$ $(C0)_{hex}$ $(01)_{hex}$
$(FA)_{hex}$ $(0F)_{hex}$ $(CE)_{hex}$ $(4F)_{hex}$ $(1E)_{hex}$ $(C6)_{hex}$ $(A3)_{hex}$ $(B8)_{hex}$ $(F4)_{hex}$ $(12)_{hex}$
$(7F)_{hex}$ $(F9)_{hex}$ $(7D)_{hex}$ $(6F)_{hex}$ $(E0)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$

There are 251 codewords in the information codeword stream (K.3.2).

According to Annex B, there are 3 blocks in Version 17 with error correction level K2, and the error correction characteristics of each blocks are:

(100,70,30), (100,70,30) (99,69,30)

The information codewords shall be divided into 3 blocks, which contain 70, 70 and 69 codewords. Utilising the Reed-Solomon error correction algorithm described in 5.5 in each block, the error correction codewords shall be generated. According to the 5.7, re-assemble information codewords and error correction codewords to the final data codeword stream (K.3.3):

$(27)_{hex}$ $(38)_{hex}$ $(C3)_{hex}$ $(0A)_{hex}$ $(35)_{hex}$ $(F9)_{hex}$ $(CF)_{hex}$ $(99)_{hex}$ $(92)_{hex}$ $(F9)_{hex}$
$(26)_{hex}$ $(A3)_{hex}$ $(E7)_{hex}$ $(3E)_{hex}$ $(76)_{hex}$ $(C9)_{hex}$ $(AE)_{hex}$ $(A3)_{hex}$ $(7F)_{hex}$ $(9C)_{hex}$
$(FA)_{hex}$ $(9C)_{hex}$ $(B5)_{hex}$ $(F9)_{hex}$ $(CF)_{hex}$ $(86)_{hex}$ $(F9)_{hex}$ $(CF)_{hex}$ $(93)_{hex}$ $(E3)_{hex}$
$(1A)_{hex}$ $(3E)_{hex}$ $(73)_{hex}$ $(E0)_{hex}$ $(80)_{hex}$ $(04)_{hex}$ $(5F)_{hex}$ $(9C)_{hex}$ $(F8)_{hex}$ $(10)_{hex}$
$(FE)_{hex}$ $(B3)_{hex}$ $(E0)_{hex}$ $(00)_{hex}$ $(FA)_{hex}$ $(DF)_{hex}$ $(82)_{hex}$ $(00)_{hex}$ $(11)_{hex}$ $(7F)_{hex}$
$(47)_{hex}$ $(A1)_{hex}$ $(FF)_{hex}$ $(F2)_{hex}$ $(1B)_{hex}$ $(F4)_{hex}$ $(D5)_{hex}$ $(9F)_{hex}$ $(FF)_{hex}$ $(21)_{hex}$
$(BF)_{hex}$ $(49)_{hex}$ $(04)_{hex}$ $(C9)_{hex}$ $(4F)_{hex}$ $(FF)_{hex}$ $(20)_{hex}$ $(01)_{hex}$ $(FD)_{hex}$ $(87)_{hex}$

$(0E)_{hex}$ $(05)_{hex}$ $(48)_{hex}$ $(1C)_{hex}$ $(F0)_{hex}$ $(5C)_{hex}$ $(DC)_{hex}$ $(E0)_{hex}$ $(68)_{hex}$ $(ED)_{hex}$
$(BF)_{hex}$ $(45)_{hex}$ $(73)_{hex}$ $(5F)_{hex}$ $(DC)_{hex}$ $(8A)_{hex}$ $(F2)_{hex}$ $(76)_{hex}$ $(28)_{hex}$ $(CF)_{hex}$
$(11)_{hex}$ $(94)_{hex}$ $(AB)_{hex}$ $(85)_{hex}$ $(E8)_{hex}$ $(DE)_{hex}$ $(93)_{hex}$ $(29)_{hex}$ $(96)_{hex}$ $(29)_{hex}$

$(85)_{hex}$ $(7F)_{hex}$ $(FF)_{hex}$ $(20)_{hex}$ $(00)_{hex}$ $(FD)_{hex}$ $(0B)_{hex}$ $(1F)_{hex}$ $(FF)_{hex}$ $(C9)_{hex}$
$(95)_{hex}$ $(BF)_{hex}$ $(6A)_{hex}$ $(1F)_{hex}$ $(6A)_{hex}$ $(B6)_{hex}$ $(82)_{hex}$ $(99)_{hex}$ $(1A)_{hex}$ $(87)_{hex}$
$(36)_{hex}$ $(70)_{hex}$ $(6C)_{hex}$$(A0)_{hex}$ $(D9)_{hex}$ $(D1)_{hex}$ $(B3)_{hex}$ $(FF)_{hex}$ $(FF)_{hex}$ $(E5)_{hex}$
$(F5)_{hex}$ $(BF)_{hex}$ $(04)_{hex}$ $(00)_{hex}$ $(22)_{hex}$ $(FE)_{hex}$ $(E0)_{hex}$ $(5C)_{hex}$ $(01)_{hex}$ $(21)_{hex}$
$(BF)_{hex}$ $(70)_{hex}$ $(38)_{hex}$ $(C8)_{hex}$ $(90)_{hex}$ $(DF)_{hex}$ $(B8)_{hex}$ $(1C)_{hex}$ $(8E)_{hex}$ $(48)_{hex}$
$(10)_{hex}$ $(FF)_{hex}$ $(70)_{hex}$ $(36)_{hex}$ $(CC)_{hex}$ $(B8)_{hex}$ $(1C)_{hex}$ $(38)_{hex}$ $(5C)_{hex}$ $(10)_{hex}$
$(30)_{hex}$ $(2e)_{hex}$ $(07)_{hex}$ $(0A)_{hex}$ $(97)_{hex}$ $(03)_{hex}$ $(22)_{hex}$ $(0B)_{hex}$ $(81)_{hex}$ $(60)_{hex}$ (K.3.3)

$(A9)_{hex}$ $(B6)_{hex}$ $(77)_{hex}$ $(AB)_{hex}$ $(56)_{hex}$ $(CC)_{hex}$ $(27)_{hex}$ $(9D)_{hex}$ $(BD)_{hex}$ $(8C)_{hex}$
$(C6)_{hex}$ $(9F)_{hex}$ $(AA)_{hex}$ $(8D)_{hex}$ $(9A)_{hex}$ $(D4)_{hex}$ $(2D)_{hex}$ $(F0)_{hex}$ $(0D)_{hex}$ $(7A)_{hex}$
$(81)_{hex}$ $(A6)_{hex}$ $(21)_{hex}$ $(AA)_{hex}$ $(42)_{hex}$ $(A5)_{hex}$ $(54)_{hex}$ $(4F)_{hex}$ $(6D)_{hex}$ $(D9)_{hex}$

$(F9)_{hex}$ $(C0)_{hex}$ $(E5)_{hex}$ $(82)_{hex}$ $(E0)_{hex}$ $(7D)_{hex}$ $(AD)_{hex}$ $(70)_{hex}$ $(3C)_{hex}$ $(E8)_{hex}$
$(17)_{hex}$ $(D5)_{hex}$ $(7C)_{hex}$ $(10)_{hex}$ $(00)_{hex}$ $(8B)_{hex}$ $(FA)_{hex}$ $(3D)_{hex}$ $(0F)_{hex}$ $(FF)_{hex}$
$(90)_{hex}$ $(DF)_{hex}$ $(A6)_{hex}$ $(AC)_{hex}$ $(FF)_{hex}$ $(F9)_{hex}$ $(0D)_{hex}$ $(FA)_{hex}$ $(1E)_{hex}$ $(85)_{hex}$
$(C1)_{hex}$ $(DA)_{hex}$ $(57)_{hex}$ $(FF)_{hex}$ $(90)_{hex}$ $(21)_{hex}$$(FD)_{hex}$ $(67)_{hex}$ $(00)_{hex}$ $(01)_{hex}$
$(FA)_{hex}$ $(0F)_{hex}$ $(CE)_{hex}$ $(4F)_{hex}$ $(1E)_{hex}$ $(C6)_{hex}$ $(A3)_{hex}$ $(B8)_{hex}$ $(F4)_{hex}$ $(12)_{hex}$
$(7F)_{hex}$ $(F9)_{hex}$ $(7D)_{hex}$ $(6F)_{hex}$ $(E0)_{hex}$$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$
$(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(00)_{hex}$ $(0B)_{hex}$
$(7E)_{hex}$ $(F1)_{hex}$ $(33)_{hex}$ $(0D)_{hex}$ $(51)_{hex}$ $(EA)_{hex}$ $(93)_{hex}$ $(54)_{hex}$ $(BD)_{hex}$ $(62)_{hex}$
$(B3)_{hex}$ $(68)_{hex}$ $(17)_{hex}$ $(58)_{hex}$ $(C0)_{hex}$ $(E0)_{hex}$ $(E1)_{hex}$ $(A9)_{hex}$ $(03)_{hex}$ $(C1)_{hex}$
$(0A)_{hex}$$(C8)_{hex}$ $(72)_{hex}$ $(D2)_{hex}$ $(68)_{hex}$ $(C1)_{hex}$ $(14)_{hex}$ $(9E)_{hex}$ $(43)_{hex}$

The length of the final data codeword stream (K.3.3) is 299 codewords.

Step 4. Function Pattern placement

According to 5.8.2, Function Pattern placement for the symbol of Version 17 is shown in Figure K.17.

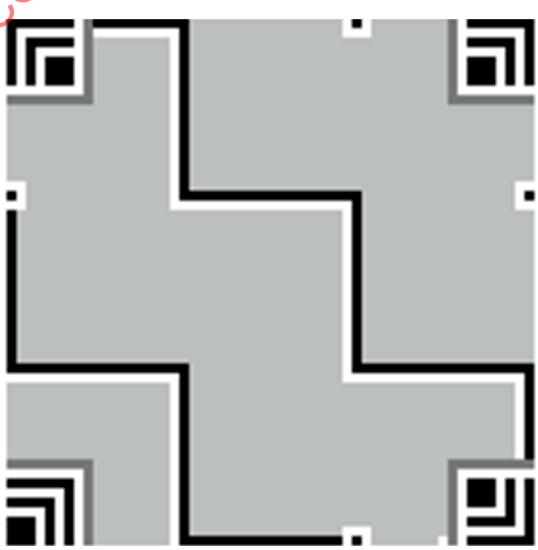


Figure K.17 — Function Pattern placement

Key

dark module

light module

function information region

information encoding region

Step 5.Data placement

The data codeword stream (K.3.3) shall be broken up using the algorithm described in 5.8.3, and hence re-assembled to the data bit stream (K.3.4) with 2392 bits:

00100111 00111000 11000011 00001010 00110101 11111001 11001111
10011001 10010010 11111001 00100110 10100011 11100111 00111110
01110110 11001001 10101110 10100011 01111111 10011100 11111010
10011100 10110101 11111001 11001111 10000110 11111001 11001111
10010011 11100011 00011010 00111110 01110011 11100000 10000000
00000100 01011111 10011100 11111000 00010000 11111110 10110011
11100000 00000000 11111010 11011111 10000010 00000000 00010001
01111111 01000111 10100001 11111111 11110010 00011011 11110100
11010101 10011111 11111111 00100001 10111111 01001001 00000100
11001001 01001111 11111111 00100000 00000001 11111101 10000111
00001110 00000101 01001000 00011100 11110000 01011100 11011100
11100000 01101000 11101101 10111111 01000101 01110011 01011111
11011100 10001010 11110010 01110110 00100000 11001111 00010001
10010100 10101011 10000101 11101000 11011110 10010011 00101001
10010110 00101001 10000101 01111111 11111111 00100000 00000000
11111101 00001011 00011111 11111111 11001001 10010101 10111111
01101010 00011111 01101010 10110110 10000010 10011001 00011010
10000111 00110110 01110000 01101100 10100000 11011001 11010001
10110011 11111111 11111111 11100101 11110101 10111111 00000100
00000000 00100010 11111110 11100000 01011100 00000001 00100001
10111111 01110000 00111000 11001000 10010000 11011111 10111000 (K.3.4)
00011100 10001110 01001000 00010000 11111111 01110000 00110110
11001100 10111000 00011100 00111000 01011100 00010000 00110000
00101110 00000111 00001010 10010111 00000011 00100010 00001011
10000001 01100000 10101001 10110110 01110111 10101011 01010110
11001100 00100111 10011101 10111101 10001100 11000110 10011111
10101010 10001101 10011010 11010100 00101101 11110000 00001101
01111010 10000001 10100110 00100001 10101010 01000010 10100101
01010100 01001111 01101101 11011001 11111001 11000000 11100101
10000010 11100000 01111101 10101101 01110000 00111100 11101000
00010111 11010101 01111100 00010000 00000000 10001011 11111010
00111101 00001111 11111111 10010000 11011111 10100110 10101100
11111111 11111001 00001101 11111010 00011110 10000101 11000001
11011010 01010111 11111111 10010000 00100001 11111101 01100111
11000000 00000001 11111010 00001111 11001110 01001111 00011110
11000110 10100011 10111000 11110100 00010010 01111111 11111001
01111101 01101111 11100000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00001011 01111110 11110001 00110011
00001101 01010001 11101010 10010011 01010100 10111101 01100010
10110011 01101000 00010111

The result after data placement is shown as Figure K.18.

Figure K.18 — Han Xin Code symbol Version 17 after data placement (K.2)

Step 6. Masking

Utilising the masking patterns in 5.8.4 to mask Figure K.18. See Figure K.19 to Figure K.22.

Figure K.19 — Masking result for Figure K.18 with masking pattern 00

Figure K.20 — Masking result for Figure K.18 with masking pattern 01

Figure K.21 — Masking result for Figure K.18 with masking pattern 10

Figure K.22 — Masking result for Figure K.18 with masking pattern 11

Calculate the penalty scores of the patterns in accordance with Table K.8.

Table K.8 — Penalty scores of masking patterns

Condition of masking solution	Data mask pattern reference for Han Xin Code	Penalty score
Non-masking	00	17754
(i+j) mod 2=0	01	18102
((i+j)mod 3+(j mod 3)) mod 2=0	10	17160
(i mod j +j mod i + i mod 3+ j mod 3) mod 2=0	11	18386

The masking solution of this Han Xin Code symbol shall be 10.

Step 7. Generation of the function information

The function information encoding is illustrated in Table K.9.

Table K.9 — Encoding of function information

	Encoding for function information of the Han Xin Code symbol
Version	20+17=37, encoded as 0010 0101
Error correction level	01
Masking solution	10

Utilising the algorithm in Annex E, generate the function information (K.3.5):

0010 0101 0110 0110 0000 1011 1100 (K.3.5)

Step 8. Function information placement

Utilising the algorithm in 5.8.5, place the function information in the function information region, and get the final Han Xin Code symbol of this example (see Figure K.23).

Figure K.23 — Final Han Xin Code symbol (K.2)

Annex L
(informative)

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 any special features encountered in the symbol. This information is not encoded in the bar code symbol, but should be generated by the reader after decoding and transmitted as a prefix to the data message.

The symbology identifier string for Han Xin Code is:

]hm

where

] : is the symbology identifier flag character (ASCII value 93, Hex value 0x5D);

h : is the symbology identifier character for Han Xin Code (ASCII value 104, Hex value 0x68);

m : is a modifier character with one of the values defined in Table L.1.

Table L.1 — Han Xin Code symbology identifier option values

Modifier character value	Option	Description
0	"Generic" data, no special features set	No special features are set. The transmitted data does not follow the AIM ECI protocol
1	ECI protocol enabled	There is at least one ECI mode encoded. The transmitted data shall follow the AIM ECI protocol.
2	GS1 mode	Han Xin Code is used for representing GS1 data.
3	Reserved	
4	URI mode	Han Xin Code is used for representing URI.
5	Reserved	
6	Reserved	
7	Reserved	
8	Unicode mode	Han Xin Code is used for representing data by Unicode mode.
Other	Reserved for future use	

Annex M
(normative)

Charsets of URI mode

Table M.1 — URI-A charset

Character/URI fragment	Encoding value	Encoding (bits)
a	0	000000
b	1	000001
c	2	000010
d	3	000011
e	4	000100
f	5	000101
g	6	000110
h	7	000111
i	8	001000
j	9	001001
k	10	001010
l	11	001011
m	12	001100
n	13	001101
o	14	001110
p	15	001111
q	16	010000
r	17	010001
s	18	010010
t	19	010011
u	20	010100
v	21	010101
w	22	010110
x	23	010111
y	24	011000
z	25	011001
0	26	011010
1	27	011011
2	28	011100
3	29	011101
4	30	011110
5	31	011111
6	32	100000
7	33	100001
8	34	100010

Table M.1 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
9	35	100011
.	36	100100
/	37	100101
-	38	100110
_	39	100111
~	40	101000
:	41	101001
@	42	101010
?	43	101011
#	44	101100
=	45	101101
+	46	101110
$	47	101111
&	48	110000
http://	49	110001
https://	50	110010
ftp://	51	110011
mailto:	52	110100
ldap://	53	110101
tel:	54	110110
urn:	55	110111
www.	56	111000
.com	57	111001
.net	58	111010
.gov	59	111011
.org	60	111100
.cn	61	111101
Jump to URI-B	62	111110
Terminator of URI-A	63	111111

Table M.2 — URI-B charset

Character/URI fragment	Encoding value	Encoding (bits)
A	0	000000
B	1	000001
C	2	000010
D	3	000011
E	4	000100
F	5	000101
G	6	000110
H	7	000111
I	8	001000

Table M.2 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
J	9	001001
K	10	001010
L	11	001011
M	12	001100
N	13	001101
O	14	001110
P	15	001111
Q	16	010000
R	17	010001
S	18	010010
T	19	010011
U	20	010100
V	21	010101
W	22	010110
X	23	010111
Y	24	011000
Z	25	011001
!	26	011010
*	27	011011
(	28	011100
)	29	011101
,	30	011110
{	31	011111
}	32	100000
|	33	100001
\	34	100010
^	35	100011
[	36	100100
]	37	100101
`	38	100110
<	39	100111
>	40	101000
%	41	101001
"	42	101010
;	43	101011
.htm	44	101100
.html	45	101101
.asp	46	101110
.aspx	47	101111
.php	48	110000
.jsp	49	110001
gtin	50	110010
ser	51	110011

Table M.2 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
bat	52	110100
exp	53	110101
search	54	110110
id	55	110111
.jp	56	111000
.it	57	111001
.de	58	111010
.br	59	111011
.fr	60	111100
gs1	61	111101
Jump to URI-A	62	111110
Terminator of URI-B	63	111111

Table M.3 — URI-C charset

Character/URI fragment	Encoding value	Encoding (bits)
A	0	0000000
B	1	0000001
C	2	0000010
D	3	0000011
E	4	0000100
F	5	0000101
G	6	0000110
H	7	0000111
I	8	0001000
J	9	0001001
K	10	0001010
L	11	0001011
m	12	0001100
N	13	0001101
O	14	0001110
P	15	0001111
Q	16	0010000
R	17	0010001
S	18	0010010
T	19	0010011
U	20	0010100
V	21	0010101
w	22	0010110
X	23	0010111
Y	24	0011000
Z	25	0011001

Table M.3 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
A	26	0011010
B	27	0011011
C	28	0011100
D	29	0011101
E	30	0011110
F	31	0011111
G	32	0100000
H	33	0100001
I	34	0100010
J	35	0100011
K	36	0100100
L	37	0100101
M	38	0100110
N	39	0100111
O	40	0101000
P	41	0101001
Q	42	0101010
R	43	0101011
S	44	0101100
T	45	0101101
U	46	0101110
V	47	0101111
W	48	0110000
X	49	0110001
Y	50	0110010
Z	51	0110011
0	52	0110100
1	53	0110101
2	54	0110110
3	55	0110111
4	56	0111000
5	57	0111001
6	58	0111010
7	59	0111011
8	60	0111100
9	61	0111101
$	62	0111110
-	63	0111111
_	64	1000000
.	65	1000001
+	66	1000010
!	67	1000011
*	68	1000100

Table M.3 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
(	69	1000101
)	70	1000110
,	71	1000111
{	72	1001000
}	73	1001001
|	74	1001010
\	75	1001011
^	76	1001100
~	77	1001101
[	78	1001110
]	79	1001111
'	80	1010000
<	81	1010001
>	82	1010010
#	83	1010011
%	84	1010100
"	85	1010101
;	86	1010110
/	87	1010111
?	88	1011000
:	89	1011001
@	90	1011010
&	91	1011011
=	92	1011100
http://	93	1011101
https://	94	1011110
ftp://	95	1011111
mailto:	96	1100000
ldap://	97	1100001
tel:	98	1100010
urn:	99	1100011
www.	100	1100100
.com	101	1100101
.net	102	1100110
.gov	103	1100111
.org	104	1101000
.cn	105	1101001
.htm	106	1101010
.html	107	1101011
.asp	108	1101100
.aspx	109	1101101
.php	110	1101110
.jsp	111	1101111

Table M.3 *(continued)*

Character/URI fragment	Encoding value	Encoding (bits)
gtin	112	1110000
ser	113	1110001
bat	114	1110010
exp	115	1110011
search	116	1110100
id	117	1110101
.jp	118	1110110
.it	119	1110111
.de	120	1111000
.br	121	1111001
.fr	122	1111010
gs1	123	1111011
search	124	1111100
Jump to URI-A	125	1111101
Jump to URI-B	126	1111110
Terminator of URI-C	127	1111111

Annex N
(normative)

Source codes for Unicode modein C programming

N.1 Source codes of encoding process

```
/*
**********************************************************************************
**********
*
*   Copyright (c) Article Numbering Center of China (ANCC), 1988-2016. All rights
reserved.
*
*   The source codes of this software is the C programing implement Unicode Mode of Han
Xin Code.
*   All of software copyrights and the intellectual property rights of the source codes
belong
*   to Article Numbering Center of China (ANCC). The software copyrights and the
intellectual
*   property rights of the source codes are protected by the laws of The People's Republic
of
*   China and international treaties.
*
*   Article Numbering Center of China (ANCC) allows any company, person and institute use
the
*   source codes of this software under The MIT License (MIT) shown as follows:
*
*   Permission is hereby granted, free of charge, to any person obtaining a copy of this
software
*   and associated documentation files (the "Software"), to deal in the Software without
restriction,
*   including without limitation the rights to use, copy, modify, merge, publish,
distribute,
*   sublicense, and/or sell copies of the Software, and to permit persons to whom the
Software
*   is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included in all copies
or
*   substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING
*   BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
AND
*   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM,
*   DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
*   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*
**********************************************************************************
**********
*
*   FILE:     hanxincode_UnicodeMode_encode.h
*   VERSION:  1.2.0
*   DATE:     2018/09/01
*   AUTHOR:   Shi Yu
*
**********************************************************************************
**********
*/
```

```
#include <stdlib.h>
#include <string.h>
#include <memory.h>

#define IN
#define OUT

#ifndef BYTE
#define BYTE unsigned char
#endif
#define MAX_BIT_STREAM_LENGTH 1024000
#define MAX_DATA_LENGTH 1024000

//Structure used for byte pattern analysis
#ifndef BytePattern_h
#define BytePattern_h
typedef struct __BytePattern
{
   int byteCount;
   BYTE minBytes[4];
   BYTE diffs[4];
   int bits[4];

   int length;
} BytePattern;
#endif

#ifndef HanXinCode_UnicodeMode_Common_h
#define HanXinCode_UnicodeMode_Common_h

#define hanxincode_UnicodeMode_1Byte_Jump 14
#define hanxincode_UnicodeMode_End        15

#endif

//Functions used, implement after the main Encoding function
#ifndef HanXinCode_UnicodeMode_Encode_Internal_h
#define HanXinCode_UnicodeMode_Encode_Internal_h

int detection_byte_pattern_for_start_positions(BYTE * data, int length, BytePattern *
patternDetected, int detect_length_for4Bytes, int detect_length_for3Bytes, int detect_
length_for2Bytes);

int calculate_encoding_bits_for_bytes_pattern(BytePattern pattern);

int calculate_encoding_bits_for_jump_pattern(int length);

int detect_byte_pattern(BYTE * data, int length, BytePattern ** p_patterns, int * p_
pattern_length);

int calculate_encoding_bits_for_patterns(BytePattern * patterns, int pattern_length);

#endif

#ifndef hanxincode_dataCompression_encode_h
#define hanxincode_dataCompression_encode_h
/*
**********************************************************************************
**********
*
* FUNCTION: hanxincode_Unicode_encode
*
*  PARAMETERS:
*   Input     (BYTE *)        data:                Input data for encoding.
*   Input     (int)           length:              The length of data.
*   Output    (BYTE  **)      p_bit_stream:        Output bit stream.
*   Output    (int *)         p_bit_stream_length: Ouput length of bit stream.
*  RETURN VALUE:  (int)
*     Result of encoding process.
*       If encoding process successes, the result will be 0. otherwise, failed.
*  DATE:     2018/09/01
```

```
*  AUTHORS:  Shi Yu
*  VERSION:  1.2
*
*********************************************************************************************
**********
*/
int hanxincode_Unicode_encode(
      IN BYTE * data,
      IN int length,
      OUT BYTE ** p_bit_stream,
      OUT int * p_bit_stream_length
      )
{
   int return_code = 0;

   BYTE * bit_stream = NULL;
   int bit_stream_length = 0;

   int step_result = 0;

   BytePattern * patterns = NULL;
   int pattern_length = 0;

   int pos = 0;

   int iiByte = 0;

   int ii, jj, bits;

   int iiPattern = 0;
   int bitsForPattern = 0;
   int bitsFor1ByteJumpPattern = 0;
   int byte_length = 0;

   int pos_byte = 0;

   int encoding = 0;

   //int byte_count = -1;

   if (NULL == data)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (length < 0)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (NULL == p_bit_stream)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (NULL == p_bit_stream_length)
   {
      return_code = -1;
      goto exit_flag;
   }

   //Step 1.Analysis Encoding data:
   step_result = detect_byte_pattern(data, length, &patterns, &pattern_length);
   if (0 != step_result)
   {
      return_code = -2;
      goto exit_flag;
   }

   //Step 2.1. Add the Indicator of Unicode mode of Han Xin Code (1001)bin as the
beginning
```

```
   bit_stream_length = calculute_encoding_bits_for_patterns(patterns, pattern_length);
   bit_stream_length = 2 * bit_stream_length;
   bit_stream = (BYTE *)malloc(bit_stream_length * sizeof(BYTE));
   if (!bit_stream)
   {
      return_code = -2;
      goto exit_flag;
   }
   //bit_stream_length = 0;
   bit_stream[0] = 1;
   bit_stream[1] = 0;
   bit_stream[2] = 0;
   bit_stream[3] = 1;
   pos = 4;
   //Step 2.2.For each data sequence of byte pattern:
   pos_byte = 0;
   for (iiPattern = 0; iiPattern < pattern_length; ++iiPattern)
   {
      if ((1 == patterns[iiPattern].byteCount) && (patterns[iiPattern].bits[0] >= 8) &&
(patterns[iiPattern].length <= 15))
      {
         //Pattern
         encoding = hanxincode_UnicodeMode_1Byte_Jump;
         bits = 4;
         for (ii = bits - 1; ii >= 0; --ii)
         {
            bit_stream[pos + ii] = encoding & 1;
            encoding = (encoding >> 1);
         }
         pos += bits;
         //Counter
         encoding = patterns[iiPattern].length;
         bits = 4;
         for (ii = bits - 1; ii >= 0; --ii)
         {
            bit_stream[pos + ii] = encoding & 1;
            encoding = (encoding >> 1);
         }
         pos += bits;
         //Data
         for (jj = 0; jj < patterns[iiPattern].length; ++jj)
         {
            if (pos_byte >= length)
            {
               return_code = -3;
               goto exit_flag;
            }
            encoding = data[pos_byte];
            pos_byte++;
            bits = 8;
            for (ii = bits - 1; ii >= 0; --ii)
            {
               bit_stream[pos + ii] = encoding & 1;
               encoding = (encoding >> 1);
            }
            pos += bits;
         }
      }
      else
      {
         //Pattern
         encoding = patterns[iiPattern].byteCount;
         bits = 4;
         for (ii = bits - 1; ii >= 0; --ii)
         {
            bit_stream[pos + ii] = encoding & 1;
            encoding = (encoding >> 1);
         }
         pos += bits;
         //Counter
         encoding = patterns[iiPattern].length;
```

```
if ((encoding >= 0) && (encoding <= 7))
{
   bits = 4 - 1;
   bit_stream[pos + 0] = 0;
   pos += 1;
}
else if ((encoding >= 8) && (encoding <= 63))
{
   bits = 8 - 2;
   bit_stream[pos + 0] = 1;
   bit_stream[pos + 1] = 0;
   pos += 2;
}
else if ((encoding >= 64) && (encoding <= 511))
{
   bits = 12 - 3;
   bit_stream[pos + 0] = 1;
   bit_stream[pos + 1] = 1;
   bit_stream[pos + 2] = 0;
   pos += 3;
}
else if ((encoding >= 512) && (encoding <= 4095))
{
   bits = 16 - 4;
   bit_stream[pos + 0] = 1;
   bit_stream[pos + 1] = 1;
   bit_stream[pos + 2] = 1;
   bit_stream[pos + 3] = 0;
   pos += 4;
}
else if ((encoding >= 4096) && (encoding <= 32767))
{
   bits = 20 - 5;
   bit_stream[pos + 0] = 1;
   bit_stream[pos + 1] = 1;
   bit_stream[pos + 2] = 1;
   bit_stream[pos + 3] = 1;
   bit_stream[pos + 4] = 0;
   pos += 5;
}
else
{
   return_code = -3;
   goto exit_flag;
}
for (ii = bits - 1; ii >= 0; --ii)
{
   bit_stream[pos + ii] = encoding & 1;
   encoding = (encoding >> 1);
}
pos += bits;
//Diff Bits
for (iiByte = 0; iiByte < patterns[iiPattern].byteCount; ++iiByte)
{
   encoding = patterns[iiPattern].bits[iiByte];
   bits = 4;
   for (ii = bits - 1; ii >= 0; --ii)
   {
      bit_stream[pos + ii] = encoding & 1;
      encoding = (encoding >> 1);
   }
   pos += bits;
}
//Min Bytes
for (iiByte = 0; iiByte < patterns[iiPattern].byteCount; ++iiByte)
{
   if (patterns[iiPattern].bits[iiByte] < 8)
   {
      encoding = patterns[iiPattern].minBytes[iiByte];
      bits = 8;
      for (ii = bits - 1; ii >= 0; --ii)
```

```
                {
                    bit_stream[pos + ii] = encoding & 1;
                    encoding = (encoding >> 1);
                }
                pos += bits;
            }
        }
        //Data
        for (jj = 0; jj < patterns[iiPattern].length; ++jj)
        {
            for (iiByte = 0; iiByte < patterns[iiPattern].byteCount; ++iiByte)
            {
                //Bytes
                if (pos_byte >= length)
                {
                    return_code = -3;
                    goto exit_flag;
                }
                encoding = data[pos_byte];
                pos_byte++;

                //Bits
                bits = patterns[iiPattern].bits[iiByte];
                if (bits < 8)
                {
                    encoding -= patterns[iiPattern].minBytes[iiByte];
                    for (ii = bits - 1; ii >= 0; --ii)
                    {
                        bit_stream[pos + ii] = encoding & 1;
                        encoding = (encoding >> 1);
                    }
                    pos += bits;
                }
                else
                {
                    for (ii = bits - 1; ii >= 0; --ii)
                    {
                        bit_stream[pos + ii] = encoding & 1;
                        encoding = (encoding >> 1);
                    }
                    pos += bits;
                }
            }
        }
    }

    //Step 3.Add the mode terminator of Unicode mode of Han Xin Code - (1111)bin, at the
end of the encoding of input data.
    encoding = hanxincode_UnicodeMode_End;
    bits = 4;
    for (ii = bits - 1; ii >= 0; --ii)
    {
        bit_stream[pos + ii] = encoding & 1;
        encoding = (encoding >> 1);
    }
    pos += bits;

    bit_stream_length = pos;

exit_flag:
    if (patterns)
    {
        free(patterns);
        patterns = NULL;
    }


    if (0 == return_code)
    {
        if (*p_bit_stream)
```

```
        {
            free(*p_bit_stream);
            *p_bit_stream = NULL;
        }
        *p_bit_stream = bit_stream;
        bit_stream = NULL;
        *p_bit_stream_length = bit_stream_length;
    }
    if (bit_stream)
    {
        free(bit_stream);
        bit_stream = NULL;
    }

    return return_code;
}

#endif


/*
**********************************************************************************************
**********
*
* FUNCTION: detect_byte_pattern
*
*  PARAMETERS:
*   Input     (BYTE *)          data:              Input BYTE stream for analysis.
*   Input     (int)             length:            The length of data.
*   Output    (BytePattern **)  p_patterns:        Output analysis result.
*   Output    (int *)           p_pattern_length:  The size of analysis result.
*  RETURN VALUE:  (int)
*     Result of analysis process.
*       If analysis process successes, the result will be 0. otherwise, failed.
*  DATE:     2016/01/01
*  AUTHORS:  Shi Yu
*  VERSION:  1.1
*
**********************************************************************************************
**********
*/
#define ANALYSIS_BYTES_FOR_4_12
#define ANALYSIS_BYTES_FOR_3_9
#define ANALYSIS_BYTES_FOR_2_6
int detect_byte_pattern(IN BYTE * data, IN int length, OUT BytePattern ** p_patterns, OUT
int * p_pattern_length)
{
    int return_code = 0;

    BytePattern * patterns = NULL;
    int pattern_length = 0;

    int i = 0;
    int pos = 0;
    BytePattern pattern;
    BytePattern currentPattern;
    BytePattern detectPattern;
    BytePattern changedPattern;

    int flagNotBigBytesRepresentation = 0;
    int lengthTemp;

    int step_result = 0;

    BYTE * patternFlags = NULL;

    int bitsPrevious = 0;
    int bitsRisen = 0;
    int bitsForChangePattern = 0;

    int flagNeedChangePattern = 0;
```

```
    BYTE tempBYTE;

    int bitsChanged;
    int bitsNotChanged;

    int totalBytes = 0;
    int totalBits = 0;

    if (NULL == data)
    {
       return_code = -1;
       goto exit_flag;
    }
    if (length < 0)
    {
       return_code = -1;
       goto exit_flag;
    }
    if (NULL == p_patterns)
    {
       return_code = -1;
       goto exit_flag;
    }
    if (NULL == p_pattern_length)
    {
       return_code = -1;
       goto exit_flag;
    }

    //Step 3.1.Initially analysis the input data in accordance of the byte patterns
    //Step 3.1.1.For the beginning of the data
    step_result = detection_byte_pattern_for_start_positions(data, length, &currentPattern,
ANALYSIS_BYTES_FOR_4, ANALYSIS_BYTES_FOR_3, ANALYSIS_BYTES_FOR_2);
    if (step_result < 0)
    {
       return_code = -2;
       goto exit_flag;
    }
    else if (0 == step_result)
    {
       return_code = 0;
       goto exit_flag;
    }

    bitsRisen = 0;
    for (ii = 0; ii < currentPattern.byteCount; ++ii)
    {
       bitsRisen += currentPattern.bits[ii];
    }
    bitsRisen *= currentPattern.length;
    bitsRisen += currentPattern.byteCount * (8 + 4);



    if (patternFlags)
    {
       free(patternFlags);
       patternFlags = NULL;
    }
    patternFlags = (BYTE *)malloc(length * sizeof(BYTE));
    if (!patternFlags)
    {
       return_code = -3;
       goto exit_flag;
    }
    memset(patternFlags, 0, length * sizeof(BYTE));
    if (patterns)
    {
       free(patterns);
       patterns = NULL;
```

```
    }
    patterns = (BytePattern *)malloc(length * sizeof(BytePattern));
    if (!patterns)
    {
       return_code = -3;
       goto exit_flag;
    }
    pattern_length = 0;
    currentPattern.length = 1;
    for (pos = 0; pos < currentPattern.byteCount; ++pos)
    {
       currentPattern.diffs[pos] = 0;
       currentPattern.bits[pos] = 0;
       currentPattern.minBytes[pos] = data[pos];
       patternFlags[pos] = currentPattern.byteCount;
    }
    memcpy(&patterns[pattern_length], &currentPattern, sizeof(currentPattern));
    ++pattern_length;
    //Step 3.1.2.For next analysis position of data:
    pos = currentPattern.byteCount;
    while (pos < length)
    {
       bitsPrevious = 0;
       for (ii = 0; ii < currentPattern.byteCount; ++ii)
       {
          bitsPrevious += pattern.bits[ii];
       }
       bitsRisen = 0;
       flagNeedChangePattern = 0;
       if (pos + currentPattern.byteCount > length)
       {
          step_result = detection_byte_pattern_for_start_positions(data + pos, length
- pos, &detectPattern, ANALYSIS_BYTES_FOR_4, ANALYSIS_BYTES_FOR_3, ANALYSIS_BYTES_FOR_2);
          if (step_result <= 0)
          {
             return_code = -3;
             goto exit_flag;
          }
          flagNeedChangePattern = 1;
          memcpy(&pattern, &detectPattern, sizeof(detectPattern));
       }

       //Step 3.1.2.8.
       if (!flagNeedChangePattern)
       {
          memcpy(&pattern, &currentPattern, sizeof(currentPattern));
          for (ii = 0; ii < currentPattern.byteCount; ++ii)
          {
             if (data[pos + ii] < pattern.minBytes[ii])
             {
                tempBYTE = pattern.minBytes[ii] + pattern.diffs[ii];
                pattern.minBytes[ii] = data[pos + ii];
                pattern.diffs[ii] = tempBYTE - data[pos + ii];
             }
             else if ((tempBYTE = (data[pos + ii] - pattern.minBytes[ii])) > pattern.
diffs[ii])
             {
                pattern.diffs[ii] = tempBYTE;
             }
             while ((1 << pattern.bits[ii]) <= pattern.diffs[ii])
             {
                pattern.bits[ii] += 1;
                ++bitsRisen;
             }
          }

          if (bitsRisen > 0)
          {
             step_result = detection_byte_pattern_for_start_positions(data + pos, length
- pos, &detectPattern, ANALYSIS_BYTES_FOR_4, ANALYSIS_BYTES_FOR_3, ANALYSIS_BYTES_FOR_2);
             if (step_result <= 0)
```

```
            {
               return_code = -3;
               goto exit_flag;
            }
            if (detectPattern.byteCount != currentPattern.byteCount)
            {
               flagNeedChangePattern = 1;
               memcpy(&pattern, &detectPattern, sizeof(detectPattern));
            }
            else
            {
               bitsChanged = calculate_encoding_bits_for_bytes_pattern(currentPattern);
               bitsChanged += calculate_encoding_bits_for_bytes_pattern(detectPattern);

               changedPattern.byteCount = currentPattern.byteCount;
               changedPattern.length = currentPattern.length + detectPattern.length;
               for (ii = 0; ii < currentPattern.byteCount; ++ii)
               {
                  changedPattern.bits[ii] = (currentPattern.bits[ii] >= detectPattern.
bits[ii]) ? currentPattern.bits[ii] : detectPattern.bits[ii];
                  changedPattern.minBytes[ii] = (currentPattern.minBytes[ii] <=
detectPattern.minBytes[ii]) ? currentPattern.minBytes[ii] : detectPattern.minBytes[ii];

                  tempBYTE = currentPattern.minBytes[ii] + currentPattern.diffs[ii]
- changedPattern.minBytes[ii];
                  changedPattern.diffs[ii] = tempBYTE;
                  tempBYTE = detectPattern.minBytes[ii] + detectPattern.diffs[ii]
- changedPattern.minBytes[ii];
                  if (tempBYTE > changedPattern.diffs[ii])
                  {
                     changedPattern.diffs[ii] = tempBYTE;
                  }
                  while ((1 << changedPattern.bits[ii]) <= changedPattern.diffs[ii])
                  {
                     changedPattern.bits[ii] += 1;
                  }
               }
               bitsNotChanged = calculate_encoding_bits_for_bytes_pattern(changedPattern);

               if (bitsChanged < bitsNotChanged)
               {
                  flagNeedChangePattern = 1;
                  memcpy(&pattern, &detectPattern, sizeof(detectPattern));
               }
            }

         }


      }

      if (!flagNeedChangePattern)
      {
         currentPattern.length += 1;
         memcpy(currentPattern.minBytes, pattern.minBytes, 4 * sizeof(BYTE));
         memcpy(currentPattern.diffs, pattern.diffs, 4 * sizeof(BYTE));
         memcpy(currentPattern.bits, pattern.bits, 4 * sizeof(int));
      }
      else
      {
         currentPattern.byteCount = pattern.byteCount;
         for (ii = 0; ii < currentPattern.byteCount; ++ii)
         {
            if (pos + ii > length)
            {
               return_code = -3;
               goto exit_flag;
            }
            currentPattern.minBytes[ii] = data[pos + ii];
            currentPattern.bits[ii] = 0;
            currentPattern.diffs[ii] = 0;
```

```
            }
            currentPattern.length = 1;
            ++pattern_length;
        }
        memcpy(&patterns[pattern_length - 1], &currentPattern, sizeof(currentPattern));
        for (ii = 0; ii < currentPattern.byteCount; ++ii)
        {
            patternFlags[pos + ii] = currentPattern.byteCount;
        }
        pos += currentPattern.byteCount;
    }

    //Step 3.2.Using the initial byte pattern analysis result, optimize the byte pattern:
    flagNeedChangePattern = 1;
    while (flagNeedChangePattern)
    {
        flagNeedChangePattern = 0;

        for (pos = pattern_length - 2; pos >= 0; --pos)
        {
            memcpy(&currentPattern, &patterns[pos], sizeof(BytePattern));
            memcpy(&detectPattern, &patterns[pos + 1], sizeof(BytePattern));

            bitsNotChanged = calculute_encoding_bits_for_bytes_pattern(currentPattern);
            bitsNotChanged += calculute_encoding_bits_for_bytes_pattern(detectPattern);

            //Step 3.2.1.If the data sequence and the previous data sequence use the same
byte pattern:
            if (patterns[pos].byteCount == patterns[pos + 1].byteCount)
            {
                changedPattern.byteCount = currentPattern.byteCount;
                changedPattern.length = currentPattern.length + detectPattern.length;
                for (ii = 0; ii < currentPattern.byteCount; ++ii)
                {
                    changedPattern.bits[ii] = (currentPattern.bits[ii] >= detectPattern.
bits[ii]) ? currentPattern.bits[ii] : detectPattern.bits[ii];
                    changedPattern.minBytes[ii] = (currentPattern.minBytes[ii] <=
detectPattern.minBytes[ii]) ? currentPattern.minBytes[ii] : detectPattern.minBytes[ii];

                    tempBYTE = currentPattern.minBytes[ii] + currentPattern.diffs[ii]
- changedPattern.minBytes[ii];
                    changedPattern.diffs[ii] = tempBYTE;
                    tempBYTE = detectPattern.minBytes[ii] + detectPattern.diffs[ii]
- changedPattern.minBytes[ii];
                    if (tempBYTE > changedPattern.diffs[ii])
                    {
                        changedPattern.diffs[ii] = tempBYTE;
                    }
                    while ((1 << changedPattern.bits[ii]) <= changedPattern.diffs[ii])
                    {
                        changedPattern.bits[ii] += 1;
                    }
                }
                bitsChanged = calculute_encoding_bits_for_bytes_pattern(changedPattern);

                if (bitsChanged < bitsNotChanged)
                {
                    flagNeedChangePattern = 1;

                    if (pos + 2 < pattern_length)
                    {
                        memcpy(&patterns[pos + 1], &patterns[pos + 2], (pattern_length - pos
- 2) * sizeof(BytePattern));
                    }
                    --pattern_length;
                    memcpy(&patterns[pos], &changedPattern, sizeof(BytePattern));
                    continue;
                }
            }

    //Step 3.2.2.If total byte length of two byte patterns is less than 16:
```

```
         bitsPrevious = currentPattern.byteCount * currentPattern.length + detectPattern.
byteCount * detectPattern.length;
         if (bitsPrevious < 16)
         {
            changedPattern.byteCount = 1;
            changedPattern.length = bitsPrevious;
            changedPattern.bits[0] = 8;
            changedPattern.diffs[0] = 255;
            changedPattern.minBytes[0] = 0;

            bitsChanged = calculate_encoding_bits_for_jump_pattern(changedPattern.length);

            if (bitsChanged < bitsNotChanged)
            {
               flagNeedChangePattern = 1;
               if (pos + 2 < pattern_length)
               {
                  memcpy(&patterns[pos + 1], &patterns[pos + 2], (pattern_length - pos
- 2) * sizeof(BytePattern));
               }
               --pattern_length;
               memcpy(&patterns[pos], &changedPattern, sizeof(BytePattern));
               continue;
            }
         }
      }
   }

   //Calculate the bis will be used in this situation
   totalBytes = 0;
   totalBits = 0;
   for (ii = 0; ii < pattern_length; ++ii)
   {
      bitsChanged = calculate_encoding_bits_for_bytes_pattern(patterns[ii]);

      totalBytes += patterns[ii].byteCount * patterns[ii].length;
      totalBits += bitsChanged;
   }
   bitsNotChanged = totalBits;
   if (totalBytes != length)
   {
      return_code = -4;
      goto exit_flag;
   }


   //To see if need to be changed to a whole 1 byte pattern segment instead to reduce the
encoding bits
   pattern.byteCount = 1;
   pattern.length = length;
   pattern.minBytes[0] = (length > 0) ? data[0] : 0;
   pattern.diffs[0] = 0;
   pattern.bits[0] = 0;
   for (pos = 1; pos < length; ++pos)
   {
      if (data[pos] < pattern.minBytes[0])
      {
         tempBYTE = pattern.minBytes[0] + pattern.diffs[0];
         pattern.minBytes[0] = data[pos];
         pattern.diffs[0] = tempBYTE - data[pos];
      }
      else if ((tempBYTE = (data[pos] - pattern.minBytes[0])) > pattern.diffs[0])
      {
         pattern.diffs[0] = tempBYTE;
      }
      while ((1 << pattern.bits[0]) <= pattern.diffs[0])
      {
         pattern.bits[0] += 1;
      }
   }
   bitsChanged = calculate_encoding_bits_for_bytes_pattern(pattern);
```

```
    if (bitsChanged < bitsNotChanged)
    {
        if (patterns)
        {
            free(patterns);
            patterns = NULL;
        }
        pattern_length = 1;
        patterns = (BytePattern *)malloc(pattern_length * sizeof(BytePattern));
        if (!patterns)
        {
            return_code = -3;
            goto exit_flag;
        }
        memcpy(patterns, &pattern, sizeof(BytePattern));
    }


exit_flag:
    if (patternFlags)
    {
        free(patternFlags);
        patternFlags = NULL;
    }

    if (0 == return_code)
    {
        if (*p_patterns)
        {
            free(*p_patterns);
            *p_patterns = NULL;
        }
        *p_patterns = patterns;
        patterns = NULL;
        *p_pattern_length = pattern_length;
    }
    if (patterns)
    {
        free(patterns);
        patterns = NULL;
    }
    return return_code;
}

/*
*****************************************************************************************
**********
*
* FUNCTION: calculute_encoding_bits_for_patterns
*
*   PARAMETERS:
*    Input      (BytePattern *)   patterns:        Input byte pattern analysis result.
*    Input      (int)             pattern_length:  The length of input byte pattern analysis
result.
*   RETURN VALUE:  (int)
*      Result of encoding bits.
*        If calculutation process successes, the result will be greater or equal than 0,
*        which is the encoding bis will taken. otherwise, failed.
*   DATE:     2018/09/01
*   AUTHORS:  Shi Yu
*   VERSION:  1.2
*
*****************************************************************************************
**********
*/
int calculute_encoding_bits_for_patterns(IN BytePattern * patterns, IN int pattern_length)
{
    int result = 0;

    int length = 0;
```

```
   int ii;

   if (NULL == patterns)
   {
      goto exit_flag;
   }

   for (ii = 0; ii < pattern_length; ++ii)
   {
      result += calculate_encoding_bits_for_bytes_pattern(patterns[ii]);
   }

   result += 4 + 4;

exit_flag:
   return result;
}

/*
**********************************************************************************
**********
*
* FUNCTION: calculate_encoding_bits_for_jump_pattern
*
*  PARAMETERS:
*   Input     (int)        length:  The length for 1 byte jump pattern.
*  RETURN VALUE:  (int)
*     Result of encoding bits.
*       If calculutation process successes, the result will be greater or equal than 0,
*       which is the encoding bis will taken. otherwise, failed.
*  DATE:     2016/01/01
*  AUTHORS:  Shi Yu
*  VERSION:  1.0
*
**********************************************************************************
**********
*/
int calculate_encoding_bits_for_jump_pattern(IN int length)
{
   int step_result = 0;

   step_result = 4 + 4 + (length << 3);
exit_flag:

   return step_result;
}

/*
**********************************************************************************
**********
*
* FUNCTION: calculate_encoding_bits_for_bytes_pattern
*
*  PARAMETERS:
*   Input     (BytePattern)   pattern:  Input byte pattern.
*  RETURN VALUE:  (int)
*     Result of encoding bits.
*       If calculutation process successes, the result will be greater or equal than 0,
*       which is the encoding bis will taken. otherwise, failed.
*  DATE:     2016/01/01
*  AUTHORS:  Shi Yu
*  VERSION:  1.0
*
**********************************************************************************
**********
*/
int calculate_encoding_bits_for_bytes_pattern(IN BytePattern pattern)
{
   int step_result = 0;
```

```
    int ii;

    int startBits = 0;

    if (pattern.length < 0)
    {
        step_result = -1;
        goto exit_flag;
    }

    step_result = 0;
    for (ii = 0; ii < pattern.byteCount; ++ii)
    {
        if ((pattern.bits[ii] > 8) || (pattern.bits[ii] < 0))
        {
            step_result = -2;
            goto exit_flag;
        }
        step_result += pattern.bits[ii];
        if (8 == pattern.bits[ii])
        {
            startBits += 4;
        }
        else
        {
            startBits += 12;
        }
    }
    step_result *= pattern.length;
    step_result += startBits;

    startBits = -1;
    for (ii = 1; ii <= 8; ++ii)
    {
        //4 * ii bytes use a 4 * ii - ii bits counter
        if (pattern.length < (1 << (3 * ii)))
        {
            startBits = 4 * ii;
            break;
        }
    }
    if (startBits <= 0)
    {
        step_result = -3;
        goto exit_flag;
    }
    step_result += startBits;

    step_result += 4;
exit_flag:

    return step_result;
}

/*
*****************************************************************************************
**********
*
* FUNCTION: detection_byte_pattern_for_start_positions
*
*  PARAMETERS:
*    Input      (BYTE *)          data:                    Input BYTE stream for analysis.
*    Input      (int)             length:                  The length of data.
*    Output     (BytePattern *)   patternDetected:         Output analysis result.
*    Input      (int)             detect_length_for4Bytes: The size for analysis for 4
bytes pattern.
*    Input      (int)             detect_length_for3Bytes: The size for analysis for 3
bytes pattern.
*    Input      (int)             detect_length_for2Bytes: The size for analysis for 2
bytes pattern.
*  RETURN VALUE:  (int)
```

```
*     Result of analysis process.
*       If analysis process successes, the result will be greater than 0.
*       0 means the data is empty. otherwise, failed.
*  DATE:     2016/01/01
*  AUTHORS:  Shi Yu
*  VERSION:  1.2
*
*************************************************************************************
**********
*/
//
int detection_byte_pattern_for_start_positions(IN BYTE * data, IN int length, OUT
BytePattern * patternDetected, IN int detect_length_for4Bytes, IN int detect_length_
for3Bytes, IN int detect_length_for2Bytes)
{
   int return_code = 0;

   int byteCountMax = 0;

   int byteCount = 0;

   int ii = 0;
   int pos = 0;
   BytePattern pattern;
   float encodingEff = 0;
   BytePattern currentPattern;
   float currentEncodingEff = 0.0f;
   BytePattern previousPattern;

   BYTE tempBYTE;

   int flagNotBigBytesRepresentation = 0;
   int detect_length;

   //0.check
   if (length <= 0)
   {
      return_code = 0;
      goto exit_flag;
   }
   if (NULL == data)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (NULL == patternDetected)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (detect_length_for4Bytes <= 0)
   {
      detect_length_for4Bytes = 12;
   }
   if (detect_length_for3Bytes <= 0)
   {
      detect_length_for3Bytes = 9;
   }
   if (detect_length_for2Bytes <= 0)
   {
      detect_length_for2Bytes = 6;
   }

   //1.analysis
   previousPattern.byteCount = 0;
   for (byteCountMax = 4; byteCountMax >= 2; --byteCountMax)
   {
      switch (byteCountMax)
      {
      case 4:
         detect_length = detect_length_for4Bytes;
```

```
            break;
        case 3:
            detect_length = detect_length_for3Bytes;
            break;
        case 2:
            detect_length = detect_length_for2Bytes;
            break;
        }
        flagNotBigBytesRepresentation = 0;
        if (length < detect_length)
        {
            flagNotBigBytesRepresentation = 1;
            continue;
        }

        currentEncodingEff = (float)(detect_length << 4);
        for (byteCount = 1; byteCount <= byteCountMax; ++byteCount)
        {
            pattern.byteCount = byteCount;
            for (ii = 0; ii < pattern.byteCount; ++ii)
            {
                pattern.minBytes[ii] = data[ii];
                pattern.diffs[ii] = 0;
                pattern.bits[ii] = 0;
                pattern.length = 0;
            }
            for (pos = 0; pos + pattern.byteCount <= detect_length; pos += pattern.byteCount)
            {
                pattern.length++;
                for (ii = 0; ii < pattern.byteCount; ++ii)
                {
                    if (data[pos + ii] < pattern.minBytes[ii])
                    {
                        tempBYTE = pattern.minBytes[ii] + pattern.diffs[ii];
                        pattern.minBytes[ii] = data[pos + ii];
                        pattern.diffs[ii] = tempBYTE - data[pos + ii];
                    }
                    else if ((tempBYTE = (data[pos + ii] - pattern.minBytes[ii])) > pattern.
diffs[ii])
                    {
                        pattern.diffs[ii] = tempBYTE;
                    }
                    while ((1 << pattern.bits[ii]) <= pattern.diffs[ii])
                    {
                        pattern.bits[ii] += 1;
                    }
                }
            }
            encodingEff = 0;
            for (ii = 0; ii < pattern.byteCount; ++ii)
            {
                encodingEff += (float)pattern.bits[ii];
            }
            encodingEff *= (float)pattern.length;
            encodingEff += (float)(pattern.byteCount * (8 + 4));
            encodingEff /= (float)(pattern.byteCount * pattern.length);
            if (encodingEff <= currentEncodingEff)
            {
                currentEncodingEff = encodingEff;
                memcpy(&currentPattern, &pattern, sizeof(pattern));
            }
        }

        if (previousPattern.byteCount == currentPattern.byteCount)
        {
            currentPattern.length = previousPattern.length;
        }
        else
        {
            memcpy(&previousPattern, &currentPattern, sizeof(currentPattern));
        }
```

```
      if (currentPattern.byteCount < byteCountMax)
      {
         flagNotBigBytesRepresentation = 1;
      }

      if (!flagNotBigBytesRepresentation)
      {
         break;
      }
   }
   if (flagNotBigBytesRepresentation)
   {
      currentPattern.byteCount = 1;
      currentPattern.minBytes[0] = data[0];
      currentPattern.diffs[0] = 0;
      currentPattern.bits[0] = 0;
      if (1 == previousPattern.byteCount)
      {
         currentPattern.length = previousPattern.length;
      }
      else
      {
         currentPattern.length = 1;
      }
   }

   memcpy(patternDetected, &currentPattern, sizeof(currentPattern));
   return_code = currentPattern.byteCount;

exit_flag:
   return return_code;
}
```

N.2 Source codes of decoding process

```
/*
************************************************************************************
**********
*
*   Copyright (c) Article Numbering Center of China (ANCC), 1988-2018. All rights
reserved.
*
*   The source codes of this software is the C programing implement URI Mode of Han Xin
Code.
*   All of software copyrights and the intellectual property rights of the source codes
belong
*   to Article Numbering Center of China (ANCC). The software copyrights and the
intellectual
*   property rights of the source codes are protected by the laws of The People's Republic
of
*   China and international treaties.
*
*   Article Numbering Center of China (ANCC) allows any company, person and institute use
the
*   source codes of this software under The MIT License (MIT) shown as follows:
*
*   Permission is hereby granted, free of charge, to any person obtaining a copy of this
software
*   and associated documentation files (the "Software"), to deal in the Software without
restriction,
*   including without limitation the rights to use, copy, modify, merge, publish,
distribute,
*   sublicense, and/or sell copies of the Software, and to permit persons to whom the
Software
*   is furnished to do so, subject to the following conditions:
*
*   The above copyright notice and this permission notice shall be included in all copies
or
*   substantial portions of the Software.
*
*   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
```

```
INCLUDING
*   BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
AND
*   NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM,
*   DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM,
*   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*
**********************************************************************************************
**********
*
*   FILE:     hanxincode_dataCompression_decode.h
*   VERSION:  1.2.0
*   DATE:     2018/09/01
*   AUTHOR:   Shi Yu
*
**********************************************************************************************
**********
*/
#include <stdlib.h>
#include <string.h>
#include <memory.h>

#define IN
#define OUT

#ifndef BYTE
#define BYTE unsigned char
#endif
#define MAX_BIT_STREAM_LENGTH 1024000
#define MAX_DATA_LENGTH 1024000

//Structure used for byte pattern analysis
#ifndef BytePattern_h
#define BytePattern_h
typedef struct __BytePattern
{
   int byteCount;
   BYTE minBytes[4];
   BYTE diffs[4];
   int bits[4];

   int length;
} BytePattern;
#endif

#ifndef HanXinCode_UnicodeMode_Common_h
#define HanXinCode_UnicodeMode_Common_h

#define hanxincode_UnicodeMode_1Byte_Jump  14
#define hanxincode_UnicodeMode_End         15

#endif

#ifndef hanxincode_UnicodeMode_decode_h
#define hanxincode_UnicodeMode_decode_h
/*
**********************************************************************************************
**********
*
* FUNCTION: hanxincode_Unicode_decode
*
*  PARAMETERS:
*   Input     (BYTE *)          bit_stream:             The bit stream for decoding.
*   Input     (int)             bit_stream_length:      The length of bit stream for
decoding.
*   Output    (BYTE  **)        p_data:                 Output decoding result byte
stream.
*   Output    (int *)           p_length:               Ouput length of decoding result
```

```
byte stream.
*   Output    (int *)           p_next_read_position:   Next read position in bit stream.
*  RETURN VALUE:  (int)
*     Result of decoding process.
*       If decoding process successes, the result will be 0. otherwise, failed.
*  DATE:     2018/09/01
*  AUTHORS:  Shi Yu
*  VERSION:  1.2
*
**************************************************************************************************
**********
*/
int hanxincode_Unicode_decode(
      IN BYTE * bit_stream,
      IN int bit_stream_length,
      OUT BYTE ** p_data,
      OUT int * p_length,
      OUT int * p_next_read_position
      )
{
   int return_code = 0;

   BYTE * data = NULL;
   int length = 0;

   int pos, pos_data;

   int pos_bytePattern = 0;

   int iiByte = 0;

   int ii, jj;

   int bits = 0;

   BytePattern pattern;

   int encoding = 0;

   if (NULL == bit_stream)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (bit_stream_length < 8)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (NULL == p_data)
   {
      return_code = -1;
      goto exit_flag;
   }
   if (NULL == p_length)
   {
      return_code = -1;
      goto exit_flag;
   }

   //Check Mode Indicator
   if (1 != bit_stream[0])
   {
      return_code = -2;
      goto exit_flag;
   }
   if (0 != bit_stream[1])
   {
      return_code = -2;
      goto exit_flag;
   }
```