UNIVERSAL POSTAL UNION

Physical encoding standards

ID-tagging of letter mail items – Part C: BNB-62 encoding specification

- UPU status: **S**
- Date of adoption at this status: **16 April 2002**
- Date of approval of this version: **16 April 2002**

Disclaimer

This document contains the latest information available at the time of publication. The Universal Postal Union offers no warrants, express or implied, regarding the accuracy, sufficiency, merchantability or fitness for any purpose of the information contained herein. Any use made thereof is entirely at the risk and for the account of the user.

Warning – Intellectual Property

The Universal Postal Union draws attention to the possibility that the implementation of this standard may involve the use of a claimed intellectual property right. Recipients of this document are invited to submit, with their comments, notification of any relevant rights of which they are aware and to provide supporting documentation.

As of the date of approval of this standard, the Universal Postal Union had not received such notice of any intellectual property which may be required to implement this standard, other than what is indicated in this publication. Nevertheless, the Universal Postal Union disowns any responsibility concerning the existence of intellectual property rights of third parties, embodied fully or partly, in this Universal Postal Union Standard.

Universal Postal Union – International Bureau
Standards Programme
3000 Berne 15
SWITZERLAND
Tel: + 41 31 350 3111
Fax: + 41 31 350 3110
E-mail: standards@upu.int

Contents

Foreword

Postal services form part of the daily life of people all over the world. The Universal Postal Union (UPU) is the specialized institution of the United Nations that regulates the universal postal service. The postal services of its 189 member countries form the largest physical distribution network in the world. Some 6.2 million postal employees working in over 700 000 post offices all over the world handle an annual total of 430 billion letters, printed matter and parcels in the domestic service and almost 10 billion letters, printed matter and parcels in the international service. Keeping pace with the changing communications market, posts are increasingly using new communication and information technologies to move beyond what is traditionally regarded as their core postal business. They are meeting higher customer expectations with an expanded range of products and value-added services.

Standards are important prerequisites for effective postal operations and for interconnecting the global network. The UPU's Standards Board develops and maintains a growing number of standards to improve the exchange of postal-related information between posts, and promotes the compatibility of UPU and international postal initiatives. It works closely with posts, customers, suppliers and other partners, including various international organizations. The Standards Board ensures that coherent standards are developed in areas such as electronic data interchange (EDI), mail encoding, postal forms and meters.

UPU standards are drafted in accordance with the rules given in Part V of the "General information on UPU standards" and are published by the UPU International Bureau in accordance with Part VII of that publication.

This document forms Part C of a multi-part UPU standard, S18: ID-tagging of letter mail items. It should be read, and may only be modified, in conjunction with the main body of the standard, Part A.

S18 was originally published as a single-part standard, but has been split into parts in order to simplify the specification of different ID-tag encoding formats. The present document provides a specification of the encoding of UPU ID-tags on items using a 62-position bar-no-bar code, referred to as BNB-62, which is compliant with the ID-tag format in use by An Post (Ireland), Canada Post and USPS. It is published only to provide other parties with the specifications necessary to enable them to read ID-tags printed by these three organisations and should not be used as the basis for printing ID-tags by any other issuer.

The document was created from a specification provided by USPS. It represents an addition to S18–4 and has been given version number 5 to maintain consistency with the version numbering of the original standard.

Since this represents the first published version, there are no change marks.

Introduction

A general introduction to all parts of the standard is provided in S18a, to which the reader is referred. This part deals only with the encoding of ID-tags in the form of a 62-position bar-no-bar code, BNB-62, printed on the reverse side of items, in area R1, using fluorescent ink. It is arranged under six main headings:

- *Usage limitations – limited issuance*: explains that only designated issuers may apply BNB-62 ID-tags in accordance with this specification, though any organisation with appropriate equipment may read and use them ;
- *Value range limitations*: defines limitations on the values of data elements used in ID-tags which are to be represented on items in the form of a BNB-62 bar code;
- *Encoding specification*: specifies the construction of a 62-position bar no bar code from ID-tag data elements;
- *Printing of the bar code*: to allow the association of computer data with a physical item, the ID-tag is printed on the item itself. This section defines required ink and printing parameters;
- *Reading and interpretation of BNB-62 bar codes*: specifies the validation and error correction requirements associated with the reading of ID-tags represented using BNB-62 bar codes;
- *Conversion to the message and binary representations*: describes the correspondence between BNB-62 representation and the binary and message interchange representations defined in S18a.

The above definition is supported by an informative annex:

- *UPU ID-tag 62-position BNB bar code* template: provides a template which may be used for manual decoding of the data elements in printed BNB-62 representations of an ID-tag. Such manual decoding should be used with caution since, unless the complete bar code is read and processed through the appropriate error detection/correction algorithm, there is no certainty that the value obtained has been read correctly.

Physical encoding standards – ID-tagging of letter mail items – Part C: BNB-62 Encoding Specification

1 Scope

This part of the standard defines the representation of ID-tags as a 62-position bar-no-bar code (BNB-62) printed in fluorescent ink in area R1 on the reverse side of items.

BNB-62 encoding is one of two encoding specifications supported by this standard[1] for the printing of ID-tags in area R1, the other being BNB-78, which is specified in part B of the standard.

NOTE: Representation in the form of a 4-state code printed on the front of the item is covered in S18d.

BNB-62 encoding is authorised for use only by three issuers: An Post (Ireland), Canada Post and USPS. It should be encountered, on incoming items, only on mail items which originated in Canada, Ireland or the United States. Other issuers wishing to apply ID-tags in area R1 are required to use the BNB-78 encoding defined in S18b.

NOTE 1: ID-tags encoded in area R1 are required to be compliant with S18 (see UPU standard S19 and article RE 204 of the Letter Post Manual [2]). Effective 1 April 2003, this supports only two encodings in area R1, namely BNB-78 as defined in S18b and BNB-62 as defined herein. The latter is authorised for continued use only by the three issuers mentioned above. Where ID-tags are used, and are applied in area R1 on the reverse side of letter mail items of size up to and including C5, the use of BNB-78 encoding is mandatory, from 1 April 2003, for all other issuers.

NOTE 2: BNB-62 encoding is not considered suitable for use on flats. S18d defines a 4-state encoding which may be used for this purpose.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this standard. For references which mention specific version numbers or dates, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For references without date or version number, the latest edition of the normative document referred to applies.

See part A of the standard, S18a.

3 Terms and definitions

A number of common terms used in this document are defined in documents referred to in Normative References and in the Bibliography. Definition of frequently used or particularly important terms as well as other terms introduced in this document are given below.

See part A of the standard, S18a.

4 Symbols and abbreviations

See part A of the standard, S18a.

5 Usage limitations – limited issuance

The allocation of ID-tags and their application to items using BNB-62 representation is restricted to three issuers: An Post (Ireland), Canada Post and USPS.

---

1 References to "this standard" should be interpreted as references to S18 as a whole, not only to Part C.

NOTE 1: There are no constraints on the reading and use of ID-tags. Any mail handling organisation with appropriate reading equipment may read ID-tags on items and use these for their intended purposes. However, the encoding of ID-tags in BNB-62 format is restricted to the three identified issuers.

6 Value range limitations

In addition to the component value limitations defined in S18a, the following limitations apply to ID-tags which are to be represented in the form of BNB-62 bar codes:

NOTE: These limitations derive from the specification of BNB-62 format, which preceded the definition of UPU standard S18.

- the format identifier is limited to the value *18B;*
- only four issuer codes are supported: *CAA* for Canada Post; *IEA* for An Post and *USA* and *USB* for USPS;

  NOTE 1: In BNB-62 bar codes, the issuer code is represented by a combination of a single bit, called the C-bit, and the equipment identifier, based on the ranges specified below.

  NOTE 2: Two issuer codes are used for USPS to allow a distinction to be made between ID-tags applied by USPS itself (issuer code USA*) and ID-tags applied, under licence from USPS, by major mail producers (issuer code* USB*).*
- domain codes are not used;
- the possible values of equipment identifier are limited to:
  - *F8C-F9F*, with C-bit value *1* for An Post (issuer code IEA)
  - *E10-F8B*, with C-bit value *1* for Canada Post (issuer code CAA);
  - *001-D47*, with C-bit value *0* for USPS (issuer code USA);
  - *D48-E0F*, with C-bit value *1* for USPS-licensed mailers (issuer code USB).

  NOTE 1: In S18, equipment codes are hexadecimal. In decimal notation, these ranges correspond to 0001-3599 for USPS; 3600-3979 for Canada Post and 3980-3999 for An Post. Values of 000 *and in excess of* F9F *(decimal 3999) should not be encountered.*

  NOTE 2: Equipment codes in the range D48-E0F *(decimal 3400-3599) are allocated, by USPS to agents (mailers) which ID-tag items on USPS' behalf. Items carrying ID-tags with equipment identifiers in this range typically enter the postal system only after a delay and, if the time interval in the ID-tag is used for quality measurement purposes, should be subject to different delay criteria. In ID-tag message and binary representations, these ID-tags carry a different issued code (*USB *instead of* USA*) to permit their easy identification.*

  NOTE 3: Should it ever become necessary to licence additional issuers to use the BNB-62 representation, this would be achieved by use of C-bit value 1 in association with equipment identifiers 001-D47 and of C-bit value 0 with equipment identifiers D48-E0F. Readers should therefore be designed to accommodate this possibility.
- priority is not supported and always has value *N*.

  NOTE: The original USPS specification includes a priority-like component referred to as 'mail class'. However, this is not used in practice and the bar position concerned (the first position after the start bar) is always:
  - *space in BNB-62 ID-tags issued by USPS itself;*
  - *bar in BNB-62 ID-tags issued by An Post, Canada Post and mailers authorised to apply ID-tags by USPS.*
- the item number is limited to the range *00001-24999;*
- tracking indicator is not supported and always has value *N*.

The normal specification allows for allocation equipment with a peak processing speed of 24999 items per 30 minute time interval. In cases in which two physically distinct allocation systems each have a throughput of less than 12500 items per 30 minute period, they may share a single equipment identifier. One of the systems then uses the item number range *00001-12499*; the other uses *12500-24999*. This implies that 'equipment identifier' alone may not actually identify the unit of equipment which allocated the ID-tag.

7 Encoding specification

7.1 Data to be encoded

The ID-tag components shall be converted into an array of 13 fields, referred to as C, M_3, M_2, M_1, M_0, D_1, D_0, T_1, T_0, S_4, S_3, S_2, S_1 and S_0, as follows:

NOTE 1: This section defines the data to be included in the bar code. Because of limitations on bar code length and pitch, it is not possible to encode the ID-tag information in the same format as is used in either the message or binary representations of the ID-tag (as defined in S18a).

NOTE 2: The UPU identifier is not encoded in the BNB representations of ID-tags: given the restrictions on the encoding of area R1 defined in UPU standard S19 and in article RE 204 of the Letter Post Manual [2], it is presumed that any fluorescent BNB bar code, found in area R1, which correctly decodes in accordance with this specification or that in S18b, is a UPU ID-tag.

NOTE 3: Format identifier is not encoded in the BNB-62 representation of ID-tags: it is presumed that a fluorescent BNB bar code, found in area R1, which has approximately 62 positions and which correctly decodes in accordance with this specification, is a UPU BNB-62 ID-tag.

C: a single bit having value *0* in ID-tags issued directly by USPS (equipment identifier range *001-D47*) and value *1* in ID-tags issued by licensed US mailers, An Post and Canada Post.

M_3: the first digit of the equipment identifier expressed in decimal notation (value 0 to 3), converted to a 2-bit value using Table 1 below;

Data value to be encoded	Encoded representation in		
	4-bit field	3-bit field	2-bit field
0	1111	111	11
1	1110	110	10
2	1101	101	01
3	1100	100	00
4	1011	011	
5	1010	010	
6	1001		
7	0100		
8	0111		
9	0110		

Table 1: Data field encoding look-up table

NOTE: The 2-bit combination 00 is only used in field M_3. The possibility of its resulting in a run of more than four spaces is prevented by ensuring that it is either preceded by a bar (and followed by a maximum of two spaces) or that it is followed by a maximum of one space. For this reason, the C-bit may not have value 0 in association with equipment identifiers above E0F (3599 decimal).

*EXAMPLE: If the equipment identifier is 8E6, its decimal equivalent is 8•256 + 14•16 + 6 = 2278, which has first digit 2. **M_3** will be 01.*

M_2: the second digit of the equipment identifier expressed in decimal notation (value 0 to 9), converted to a 4-bit value using Table 1;

*EXAMPLE: If the equipment identifier is 8E6, its decimal equivalent is 8•256 + 14•16 + 6 = 2278, which has second digit 2, resulting in **M_2** being 1101.*

M_1: the third digit of the equipment identifier expressed in decimal notation (value 0 to 9), converted to a 4-bit value using Table 1;

*EXAMPLE: If the equipment identifier is 8E6, its decimal equivalent is 8•256 + 14•16 + 6 = 2278, which has third digit 7, resulting in **M_1** being 0100.*

M_0: the fourth digit of the equipment identifier expressed in decimal notation (value 0 to 9), converted to a 4-bit value using Table 1;

*EXAMPLE: If the equipment identifier is 8E6, its decimal equivalent is 8•256 + 14•16 + 6 = 2278, which has fourth digit 8, resulting in **M_0** being 0111.*

D_1: the first digit of the date within the current calendar month (value 0-3), converted to a 3-bit value using Table 1;

*EXAMPLE: If the date is the 6th, the first digit is 0, resulting in **D_1** being 111.*

D_0: the second digit of the date within the current calendar month (value 0-9), converted to a 4-bit value using Table 1;

*EXAMPLE: If the date is the 6th, the second digit is 6, resulting in **D_0** being 1001.*

NOTE: Month is not encoded in the BNB-62 representation of ID-tags: on reading of such codes, it is presumed that the calendar month is:

- *the current month if the day number in the ID-tag is less than or equal to the current day within the month;*
- *the previous calendar month otherwise.*

T_1: the first decimal digit of the number (counting from 0) of the 30 minute interval within the day during which the ID-tag was allocated, converted to a 3-bit value using Table 1;

NOTE: 30 minute intervals are counted from 00, for the time interval between 00:00 and just before 01:30, up to 47, for the time interval between 23:30 and just before 24:00. Hence, the first digit is in the range 0-4 and can be represented as a 3-bit value.

*EXAMPLE: If the ID-tag was applied at 15:23, this corresponds to the 30th half hour period (counting form 0). The first digit is 3, resulting in **T_1** being 100.*

T_0: the second decimal digit of the number of the 30 minute interval within the day, during which the ID-tag was allocated, converted to a 4-bit value using Table 1;

*EXAMPLE: If the ID-tag was applied at 15:23, this corresponds to the 30th half hour period (counting form 0). The second digit is 0, resulting in **T_0** being 1111.*

S_4: the first decimal digit of the item number, converted to a 2-bit value using Table 1;

NOTE: Item number is limited to the range 00001 to 24999, so the first digit is in the range 0-2 and can be represented as a 2-bit value.

*EXAMPLE: If the ID-tag was the 14880th to be applied in the half hour period concerned, the first digit of the serial number is 1, resulting in **S_4** being 10.*

S_3: the second decimal digit of the item number, converted to a 4-bit value using Table 1;

*EXAMPLE: If the ID-tag was the 14880th to be applied in the half hour period concerned, the second digit of the serial number is 4, resulting in **S_3** being 1011.*

S_2: the third decimal digit of the item number, converted to a 4-bit value using Table 1;

*EXAMPLE: If the ID-tag was the 14880th to be applied in the half hour period concerned, the third digit of the serial number is 8, resulting in **S_2** being 0111.*

S_1: the fourth decimal digit of the item number, converted to a 4-bit value using Table 1;

*EXAMPLE: If the ID-tag was the 14880th to be applied in the half hour period concerned, the fourth digit of the serial number is 8, resulting in **S_1** being 0111.*

S_0: the last decimal digit of the item number, converted to a 4-bit value using Table 1.

*EXAMPLE: If the ID-tag was the 14880th to be applied in the half hour period concerned, the last digit of the serial number is 0, resulting in **S_0** being 1111.*

7.2 Calculation of the error detection and correction bits

The ID-tag data shall, for the purposes of representation on the item by means of a BNB-62 bar code, be protected by the incorporation of error control codes, calculated in accordance with the following algorithm:

NOTE 1: It is not sufficient to simply record the ID-tag data content as a BNB pattern, as read errors would make the result unusable. The algorithm defined uses a polynomial code with 47 information bits and 12 redundancy bits, numbered E_{11} to E_0. A thirteenth redundancy bit, E_P, ensures that the overall bar code has even parity. When the ultimate bar code is read, these bits are used to provide an error detection and correction capability, allowing the ID-tag to be read even when it is imperfectly printed and/or partly obliterated.

NOTE 2: The specified algorithm supports correction of errors in any two bar code positions and detection of errors in three bar positions.

NOTE 3: If the number of errors exceeds the error correction capacity of the algorithm, e.g. if there are more than three bar values in error, then the specified algorithm will not result in the original bar code, but may yield an apparently valid result. This means that the algorithm is potentially sensitive to shifting of the bar code in case of erasure of or failure to recognise the start bar. In theory, this may give rise to misreads. However, these are normally detected by the validation mechanisms defined in section 9 and in the section on Reading and validation of ID-tags on items *in part A. Operational experience confirms that the BNB-62 ID-tag misread rate is low.*

NOTE 4: It should be stressed that the algorithm specified below is expressed in a simple mathematical form. Computer-coded implementations need not follow the same logic, provided that the bar pattern which results is identical to that resulting from the algorithm given here.

1. Construct a 47 element binary array F_n, with $n = 0\text{-}46$, by taking the fields defined in section 7.1 in the order of their definition and using the bits in the order left to right

2. Treating each element of F as a binary (value 0 or 1) coefficient, construct the polynomial P:

$$\sum_{n=0}^{46} F_n x^{(58-n)}$$

3. Perform modulus 2 division of P by the generator polynomial G:

$$x^{12} + x^{10} + x^8 + x^5 + x^4 + x^3 + 1$$

to give a quotient *Q* and remainder *E* such that:

$$P = Q \bullet G \oplus E$$

4. Express the coefficients of *E* as twelve elements {E_{11}, E_{10}, …, E_1, E_0} of a binary array such that:

$$E = \sum_{n=0}^{11} E_{11-n} x^{11-n}$$

NOTE: In this case, the coefficients are numbered from 11 down to 0, not (as is the case for F*) from 0 upwards.*

5. Count the number of 1 values in *P* and *E* taken together. If the total is even, set E_P to *0*; if it is odd, set E_P to *1*.

EXAMPLE: Based on the example given in section 7.1 above, the steps give:

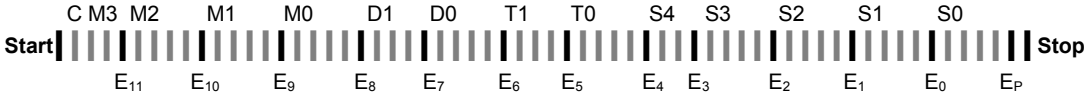
1. *F = {0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1};*
2. *P = $\{x^{56} + x^{55} + x^{54} + x^{52} + x^{50} + x^{46} + x^{45} + x^{44} + x^{43} + x^{42} + x^{41} + x^{40} + x^{37} + x^{36} + x^{33} + x^{32} + x^{31} + x^{30} + x^{29} + x^{27} + x^{25} + x^{24} + x^{22} + x^{21} + x^{20} + x^{18} + x^{17} + x^{16} + x^{15} + x^{14} + x^{13} + x^{12}\}$;*
3. *P = $\{ x^{44} + x^{43} + x^{41} + x^{38} + x^{36} + x^{34} + x^{32} + x^{31} + x^{29} + x^{28} + x^{26} + x^{21} + x^{19} + x^{17} + x^{16} + x^{15} + x^{14} + x^{12} + x^{10} + x^4 + 1\} \bullet \{x^{12} + x^{10} + x^8 + x^5 + x^4 + x^3 + 1\}$ xor $\{x^9 + x^7 + x^5 + x^3 + 1\}$;*
4. *E = {0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1};*
5. *The number of 1 values in P is 32 and the number in E is 5, giving a total of 37, which is odd. Hence E_P is 1.*

7.3 Construction of the bar code to be printed on the Item

A 62-bar position bar code shall be constructed by concatenating:

- a single start bar (**I**);
- the values of C, M_3, E_{11}, M_2, E_{10}, M_1, E_9, M_0, E_8, D_1, E_7, D_0, E_6, T_1, E_5, T_0, E_4, S_4, E_3, S_3, E_2, S_2, E_1, S_1, E_0, S_0, E_P (in that sequence), using the convention one (*1*) = bar (**I**); zero (0) = space ( );
- a single stop bar (**I**).

NOTE: The following figure shows the resulting sequence of fields (for illustration purposes not to scale and with data fields shown in grey and start, stop and error detection bits in black:



EXAMPLE:Still using the same example, the result (for illustration purposes not to scale and with bar in black and space in faint grey instead of white) will be:

Figure 1: Illustration of bar code structure, with complete example

8 Printing of the bar code

The resulting 62-position bar-no-bar code shall be encoded on the item, using fluorescent ink, in accordance with the following specifications, to be met by at least 99% of production examples.

8.1 Optical characteristics of the ink

The specifications in the S18b section on *Optical characteristics of the ink* shall be followed

8.2 Bar code characteristics

- Bar height: 4.0 ± 0.4 mm;

  NOTE: The range allows for variation in the distance between the item and the ink jet.

- Bar width (thickness): nominally 0.5 mm, with allowed variation between 0.3 mm and 0.8 mm;

  NOTE: The range allows for different absorption rates/drying times on different materials.

- Encoding density (pitch): nominal pitch of 1.66 mm (i.e. approx. 6 bars per cm). The total length of the bar code (62 bars including start and stop bar and error correction bars) shall lie within the range 9.7 to 10.7 cm;

  NOTE: The upper limit corresponds to the space available (after allowing for leading and trailing spaces) on a minimum size US envelope (i.e. 127 mm; standard letter mail has a minimum dimension of 140 mm).

- Pitch variation: not greater than 5%;

  NOTE 1: That is, the pitch may not vary by more than 5% across the whole bar code.

  NOTE 2: Notwithstanding the above printing requirement, readers should be designed to accommodate the maximum tolerance which is consistent with maintaining synchronisation, taking into account that the complete bar code should never have more than four consecutive spaces*.*

- Placement: on the reverse side of the item, in position R1 as defined by UPU Standard S19. No other fluorescent material should appear in the first 128 mm of area R1.

  NOTE 1: Position R1 is an area on the reverse side of the item, which has been reserved ***exclusively*** *for the placement of a UPU ID-tag in accordance with this standard.*

  NOTE 2: The placement of an ID-tag constructed in accordance with this standard at some other location on the item is, subject to its not contravening other UPU standards, not precluded. However, ID-tags placed in such other locations may well not be readable by parties other than the issuer.

- Vertical position:

  - equipment shall be adjusted to print the tops of the bars at a nominal position of 13 $\pm$ 1 mm from the bottom of the item;
  - on production samples, the complete bar code shall lie between parallel lines situated 4 mm and 16 mm from the bottom of the item;

  NOTE 1: The permitted range accommodates possible variations in the height of the bars and a small degree of item skew.

  NOTE 2: The specified range applies across all items and does not mean that the code on a single item can be spread over a vertical distance of 12 mm. On any single item, the skew limitations given below limit the vertical spread much more (to approx. 7.7 mm if all bars are 4 mm in height).

- Horizontal position:

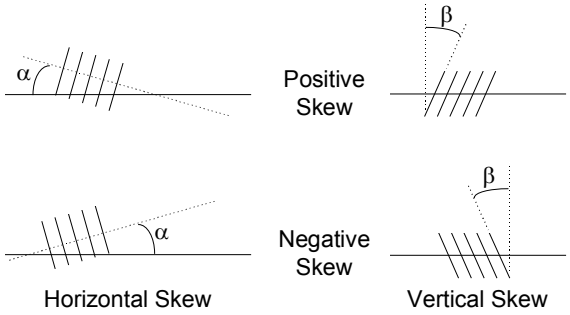
  - first (start) bar situated 15 $\pm$ 1 mm from the leading edge;
  - last (stop) bar situated at least 5 mm from the trailing edge;

  NOTE: At the maximum allowed pitch, the full 62-element code occupies 107 mm and can therefore be accommodated, with necessary leading and trailing spaces, on envelopes of 127 mm (USPS minimum) or more in length (provided that the start bar is not more than 15 mm from the leading edge on a minimum sized envelope).

- Skew:

  - the angle between the centre line of the bar code and the bottom of the mail piece shall not exceed 2° (horizontal skew);
  - the angle between each bar and the perpendicular to the centre line shall not exceed 5° (vertical skew);
  - the sum of horizontal and vertical skew shall not exceed 5°;

  NOTE: Skew limitations are illustrated in the following diagram:



$|\alpha| < 2$: $|\beta| < 5$: $|\alpha + \beta| < 5$

Figure 2: Illustration of skew limitations

9 Reading and interpretation of BNB-62 bar codes

Reading systems should be designed to discriminate between BNB-62 bar code representations of ID-tags and other fluorescent bar codes which may possibly be printed in the area being captured from. For BNB-62, this area will normally be area R1 and should be used only for ID-tag encoding in accordance with this standard. Hence, at least from 1 April 2003, it should be necessary to distinguish only between BNB-62 and the BNB-78 representation specified in S18b.

NOTE: Until April 2003, other forms of domestic ID-tag may be encountered in area R1.

Distinction between these two representations can in principle be made on the basis of the bar code pitch (1.33 mm for BNB-78 versus 1.66 mm for BNB-62) and the number of bar positions (78 versus 62).

9.1 Fully-read bar codes

A fully-read BNB-62 bar code should result in a 62-element array in which the first (start) and last (stop) elements correspond to *bar*. This array should be passed through a validation/correction process in order to detect and, if possible, correct any errors or erasures.

NOTE: The reading of bar codes is subject to error as a result of factors such as excessive skew, smudging of the fluorescent ink, obliteration or obscuration of parts of the code, etc. The ID-tag includes error correction bars, inserted in accordance with an algorithm which enables a high degree of probability of detection of any errors which occur, with a slightly lower probability of their being correctable. Prior to use of the read code, it should be processed through this algorithm in order to detect, and where possible, correct any errors which have occurred.

Annex A section A.2 informatively describes one possible validation algorithm. Bar codes which are shown to contain an error, either by the presence of invalid sequences or by failure of the validation process, should be corrected by means of an appropriate algorithm. Annex A, section A.3 provides an example (informative) implementation of such an algorithm.

9.2 Partially read bar codes

In case of damage to the bar code or the envelope, or in case of envelopes with large background variations or fluorescence in the bar code area, it is possible that the bar code reader fails to detect some bars, or detects “false” bars where there were none. The impact of this is most critical in cases in which either the start and/or stop bar are missed, or in which “false” bars are interpreted as start and/or stop bars. In both cases, the result is likely to be that the reader detects either less, or more, than 62 bar positions in the bar code.

In such cases, it may be possible to correct the bar code value by suitable pre-processing of the captured data. However, it is stressed that the results of such pre-processing should be treated with care: the error detection and correction algorithm employed is only capable of correcting up to two errors and may give erroneous results (misreads) if error correction is applied on top of pre-processing to correct for possible start / stop errors.

9.3 Checking data field values

The error detection/correction algorithm results in (eventually corrected) values for the 47 data bits making up the encoded version of the ID-tag value. These may be mapped onto the encoded data elements (see section 7.1) using reverse look-up:

EXAMPLE: The ID-tag  *may be represented, after removal of the start and stop bits, as the 60 bit binary string:*

001011010010010111011111001010011111010110110011100111111111

After re-arrangement as described in section A.2, this (with spaces inserted for convenience) becomes :

001 1101 0100 0111 111 1001 100 1111 10 1011 0111 0111 1111 001010101001 1

*This has 38 1-bits (*bars*) and the first 59 bits pass the error correction check. It has the following components:*

Component	*Position in bar code (start bar = 0)*	*Position in re-arranged bit string (counting from 0)*	*Bar value*	*Data value after reverse look up*
C	*1*	*0*	*0*	*-*
M_3	*2-3*	*1-2*	*01*	*2*
M_2	*5-8*	*3-6*	*1101*	*2*
M_1	*10-13*	*7-10*	*0100*	*7*
M_0	*15-18*	*11-14*	*0111*	*8*
D_1	*20-22*	*15-17*	*111*	*0*

D_0	24-27	18-21	1001	6
T_1	29-31	22-24	100	3
T_0	33-36	25-28	1111	0
S_4	38-39	29-30	10	1
S_3	41-44	31-34	1011	4
S_2	46-49	35-38	0111	8
S_1	51-54	39-42	0111	8
S_0	56-59	43-46	1111	0

Prior to processing the resulting data as components of an ID-tag, the following checks should be made:

1. that the error detection capability built into the encoding scheme did not reveal the presence of an uncorrected error, did not result in 'correction' of a non-existent bar and did not 'correct' more than two data bars;
2. that none of the components have bit values which do not appear in the encoding table in section 7.1;

   NOTE: For example, the value 0011 is not valid in a four bar field.
3. the components can be extracted from and parsed in accordance with the above specification for construction of the bar code;
4. that combinations of components lie within the validity range defined in section 6 and the encoding specification.

   NOTE: For example, the time interval is expressed in half hours and therefore cannot exceed 47. The combination of T_1 *equal to 4 and* T_0 *equal to 9 would thus be invalid*

If any of these checks fail, the code read from the item should not be used as an ID-tag.

NOTE 1: The error correction algorithm reliably corrects single and two-bar errors and detects odd numbers of errors. However, if the number of errors is even and greater than two, it will generate a wrong but apparently valid result. In many, though not all cases, this will be detected by the above checks.

NOTE 2: One possibility is simply that the bar code reader failed to properly read the code, due to bar detection errors caused by factors such as turned over envelope flaps or logos. Another possibility may be that the detected bar code was not, in fact, a BNB-62 encoding of a UPU ID-tag at all. Though area R1 is in principle reserved for the placement of a UPU ID-tag encoded in accordance with this standard, allowance should be made for the possibility that this area may contain either a BNB-78 code (see S18b) or may contain other information:

a) fluorescent materials which are erroneously interpreted as a bar code;

b) bar codes applied in accordance with previous (domestic ID-tagging) schemes used for video encoding purposes.

9.4 Manual interpretation of bar codes

Where necessary, BNB-62 bar codes can be read and interpreted manually, using the template in Annex B. However, care should be taken with manual interpretation since, unless the complete bar code is read and processed through the error detection/correction algorithm, there is no certainty that the value obtained has been read correctly.

NOTE: To improve the reliability of manually read ID-tags, it is recommended that the data are passed through the error detection and correction process. This can be done using a PC-based implementation of the algorithm, if the data - including the ECC data - are manually input to the PC.

10 Conversion to the message and binary representations

Conversion from the BNB-62 bar code representation to the message representation should take place by extracting the data from (eventually corrected) data fields C to S_0 in accordance with the following table (Table 2).

Data element	Message representation
UPU identifier	Always J
Format identifier	Always 18B (for BNB-62 format ID-tags)
Issuer code	Determine issuer code from the C-bit and equipment identifier (obtained as specified below):

C-bit	Equipment identifier (decimal)	Equipment identifier (hexadecimal)	Issuer Code
0	0001-3399	001-D47	USA
1	3400-3599	D48-E0F	USB
1	3600-3979	E10-F8B	CAA
1	3980-3999	F8C-F9F	IEA

NOTE: Other combinations of C-bit and equipment identifier are not supported by the current specification. However, it is recommended that readers be capable of supporting C-bit value 0 *for equipment identifiers in the range D48-E0F and of supporting C-bit value* 1 *for equipment identifiers in the range 001-D47.*

Equipment identifier	Using reverse look-up in Table 1, extract the decimal value of the equipment identifier from M_3 (bar positions 2-3) M_2 (bar positions 5-8) M_1 (bar positions 10-13) and M_0 (bar positions 15-18), with M_3 being the most significant digit. Convert the result to hexadecimal, giving a value (if the ID-tag is valid) in the range *001* to *F9F*
Item priority	always N
Serial number: generation date mmdd	Derive the calendar month number (01= January to 12=December) from the day number in the ID-tag (see below) and the current date:

Condition	Month number
ID-tag day ≤ current day	= current month
ID-tag day > current day	= current month – 1 (12 if the current month is January)

EXAMPLE:If the current date is 23 June and the ID-tag day number is 23 or less, use month 06 (June); if the ID-tag day number is 24 or more, use month 05 (May).

Using reverse look-up in Table 1, extract the tens part of day within month from D_1 (bar positions 20-22).

Using reverse look-up in Table 1, extract the units part of day within month from D_0 (bar positions 24-27).

Serial number: generation time interval hhf	Using reverse look-up in Table 1, extract the interval number (range *00-47*) from T_1 (most significant digit, in bar positions 29-31) and T_0 (bar positions 33-36) and derive hours and the first digit of minutes as:

- hh = INT(interval number/2);
- f = 0 if interval number is even; 3 if it is odd.

Serial number: item number part	Using reverse look-up in Table 1, extract the decimal value item number (range 00001-24999) from S_4 (most significant digit, in bar positions 38-39), S_3 (bar positions 41-44), S_2 (bar positions 46-49), S_1 (bar positions 51-54) and S_0 (bar positions 56-59).
Tracking indicator	Always N (no tracking).

Table 2: Conversion of BNB-62 to message representation

Conversion from the BNB-62 bar code representation to the binary representation may be performed by first converting to the message representation and then converting this to the binary form.

Annex A
(informative)

Error detection and correction algorithms

This informative Annex provides **one possible** implementation of the algorithms for calculating the error control bars and for checking and correcting bar codes read from items. It is intended **only** as an example which can be read and understood and as a possible basis for the validation of other implementations. It is almost certainly **not** the most efficient algorithm for use in any practical bar code reader. In particular, these:

- may use table look-up in place of much of the mathematics described here;
- may perform error detection and correction in one step, rather than in two steps as described in sections A.2 and A.3;
- may, due to timing restrictions, have limited capability for the pre-processing of long and short bar codes;
- may, for optimisation reasons, make use of known probability distributions in valid bar codes - for example, the dates in bar codes being processed have a high probability of being recent; except in facilities processing a very high proportion of incoming inter-administration mail, the issuer code has a high probability of being that of the processing organisation, etc.

A.1 Calculation of error control bars

The body of the standard provides a full mathematical specification of the algorithm for calculation of the error control bars. What follows here is an example C code implementation of the algorithm. Note that this is optimised for human readability, not necessarily for best performance.

```
COPYRIGHT NOTICE: This software is provided by United States Postal Service (USPS).
It can be used only for ID-tag applications which make use of ID-tags in conformity
with UPU standard S18c. The software may be freely distributed and/or modified.  A
condition of such distribution is that this copyright notice is always included in
the comments heading the software. This software is licensed "as is" without any
other warranty of any kind. USPS expressly disclaims all warranties or conditions
including, but not limited to, implied warranties or conditions of merchantability
and fitness for particular purpose and those arising by statute or otherwise in law
or from course of dealing or usage of trade. In no event shall USPS be liable for
any direct, indirect, special, incidental, consequential or other damages arising
out of this software even if USPS has been advised of the possibility of such
damages in advance.
```

```
/*
*   This module contains the function ninfo_to_bars which generates
*   an S18c format ID-tag bar code.
*
*   Principal function in this module is:
*   ninfo_to_bars(infon *info, int *bars);
*        Generate bar code with ECC, containing supplied info, being:
*        cbit:         value of the Cbit
*        machine:  machine number in range 0001-3999
*        day:      day of month in range 01-31
*        time:         time of day in half hour increments, range 00-47
*        sequence: item sequence number, range 00001-24999
*
*    Subsidiary functions are:
*        make_ecc: calculates ecc bits and parity bit
*        nbits:        sub-function of info_to_bits for single fields
*        info_to_bits: maps data onto bit values
```

```
*         bits_to_bars: maps bit values and ecc onto bar code
*
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

/* National ID Tag Data */
typedef struct
    {
    unsigned cbit;      /* Cbit */
    unsigned machine;   /* machine id, range 0001-3999 */
    unsigned day;       /* day of month, range 01-31 */
    unsigned time;      /* 00-47, half hour increments */
    unsigned sequence;  /* mailpiece sequence number, 00001-24999 */
    } infon;

/* function definitions */
void ninfo_to_bars(infon *info, int *bars);

#define GENERATOR 0x1539 /* generator polynomial x12+x10+x8+x5+x4+x3+1 */
#define GENHI     0x1000 /* the x12 term where the data goes */

/*************  DATA VARIABLES ****************/

static int bits[47];           /* data bits only */
static int parbit;             /* the parity bit */
static int this_ecc;           /* ecc bits calculated from bits[] */

/* The following codes[] table lists zero-run-length-limited codes for
 * 4-bar codes 0...9
 * 3-bar codes 0...5
 * 2-bar codes 0...3
 * 1-bar codes 0...1
 *
/*                  bit patterns                                  values    */
static unsigned codes[] = {0x0F,0x0E,0x0D,0x0C,0x0B,0x0A,0x09,/* 0-6     */
                           0x04,0x07,0x06};                   /* 7-9     */

/*************  FUNCTION DECLARATIONS ****************/

static void make_ecc(void);
static void nbits(int value,  int *field, int length);
static void info_to_bits(infon *info);
static void bits_to_bars(int *bars);

/*************  THE FUNCTIONS ***************/

static void make_ecc(void)
    /* Generate correct this_ecc and parbit matching data in bits[0..46] */
    /* Used by both ninfo_to_bars and correct */
{
int i,j;

    this_ecc = 0;
```

```
    parbit = 0;
    for (i=0; i<47; i++)
        {
         this_ecc <<= 1;                    /* left-shift one bit */
         if (bits[i])
            {
            parbit ^= 1;               /* XOR the parity bit */
             this_ecc ^= GENHI;        /* XOR the data bit */
            }
         if (this_ecc & GENHI)         /* replace 12-bit carry+data with generator */
             this_ecc ^= GENERATOR;    /* binary 010100111001, & turn off GENHI */
         }

    j = this_ecc;
    for (i=0; i<12; i++)
        {
        parbit ^= (j&1);
        j >>= 1;
        }
}

static void nbits(int value,  int *field, int length)
    /* This function fills the specified number of field bits
     * with the correct bar code for the supplied value.
     * Used by info_to_bits
     */
{
int i;
unsigned bits;

    field += (length-1);
    bits = codes[value];
    for (i=0; i<length; i++)
        {
        *field-- = bits & 1;
        bits >>= 1;
        }
}

static void info_to_bits(infon *info)
    /* Convert information values and to bar code data bits[] */
    /* Used by ninfo_to_bars */
{
int i;

    if (info->cbit)                          /* Cbit */
        bits[0] = 0;
    else
        bits[0] = 1;

    nbits(info->machine/1000, &bits[1],  2); /* machine id M3 */
    i = info->machine % 1000;
    nbits(i/100,              &bits[3],  4); /*            M2 */
    i %= 100;
    nbits(i/10,               &bits[7],  4); /*            M1 */
    nbits(i%10,               &bits[11], 4); /*            M0 */

    nbits(info->day/10,       &bits[15], 3); /* date       D1 */
    nbits(info->day%10,       &bits[18], 4); /*            D0 */

    nbits(info->time/10,      &bits[22], 3); /* time       T1 */
```

```
    nbits(info->time%10,      &bits[25], 4); /*           T0 */

    nbits(info->sequence/10000,&bits[29],2); /* sequence   S4 */
    i = info->sequence%10000;
    nbits(i/1000,             &bits[31], 4); /*           S3 */
    i %= 1000;
    nbits(i/100,              &bits[35], 4); /*           S2 */
    i %= 100;
    nbits(i/10,               &bits[39], 4); /*           S1 */
    nbits(i%10,               &bits[43], 4); /*           S0 */
}

static void bits_to_bars(int *bars)
    /* re-order bits[] to bars[] */
    /* Used by ninfo_to_bars */
{
int i;

    bars[0] = 1;                /* start */
    for (i=0; i<3; i++)
        bars[i+1] = bits[i];    /* CM3 */  /* offset = start bit */
    for (i=3; i<7; i++)
        bars[i+2] = bits[i];    /* M2 */   /* offset = start + E11 */
    for (i=7; i<11; i++)
        bars[i+3] = bits[i];    /* M1 */
    for (i=11; i<15; i++)
        bars[i+4] = bits[i];    /* M0 */
    for (i=15; i<18; i++)
        bars[i+5] = bits[i];    /* D1 */
    for (i=18; i<22; i++)
        bars[i+6] = bits[i];    /* D0 */
    for (i=22; i<25; i++)
        bars[i+7] = bits[i];    /* T1 */
    for (i=25; i<29; i++)
        bars[i+8] = bits[i];    /* T0 */
    for (i=29; i<31; i++)
        bars[i+9] = bits[i];    /* S4 */
    for (i=31; i<35; i++)
        bars[i+10] = bits[i];   /* S3 */
    for (i=35; i<39; i++)
        bars[i+11] = bits[i];   /* S2 */
    for (i=39; i<43; i++)
        bars[i+12] = bits[i];   /* S1 */
    for (i=43; i<47; i++)
        bars[i+13] = bits[i];   /* S0 */   /* offset = start + E11..E0 */

    i = this_ecc;
    bars[60] = parbit;          /* Ep */
    bars[55] = i&1;             /* E0 */
    i>>=1;
    bars[50] = i&1;             /* E1 */
    i>>=1;
    bars[45] = i&1;             /* E2 */
    i>>=1;
    bars[40] = i&1;             /* E3 */
    i>>=1;
    bars[37] = i&1;             /* E4 */
    i>>=1;
    bars[32] = i&1;             /* E5 */
    i>>=1;
    bars[28] = i&1;             /* E6 */
```

```
    i>>=1;
    bars[23] = i&1;              /* E7 */
    i>>=1;
    bars[19] = i&1;              /* E8 */
    i>>=1;
    bars[14] = i&1;              /* E9 */
    i>>=1;
    bars[ 9] = i&1;              /* E10 */
    i>>=1;
    bars[ 4] = i&1;              /* E11 */

    bars[61] = 1;                /* stop */
}

void ninfo_to_bars(infon *info, int *bars)
    /* Encode national ID Tag bar code including BCH binary ECC */
{
    info_to_bits(info);       /* convert info to data bits */
    make_ecc();               /* compute this_ecc */
    bits_to_bars(bars);       /* mix bits into bar code */
}
```

A.2 Checking the read value for errors

The following algorithm provides a method for verifying whether a fully read BNB-62 bar code has been read correctly.

1. Create a 62-element array *V*, whose first element, V_0, corresponds to the start bar and the last, V_{62}, corresponds to the stop bar and each of whose elements has value *1* if the corresponding BNB bar code position was *bar* and *0* if it was *space*. Verify that the first and last bar positions were *bar*.

   NOTE: If the first and last positions were not bar*, the bar code was not read correctly (or is not a BNB-62 ID-tag representation) and requires correction before use.*

2. Count the number *k* of *1* values in *V*;

3. Map the first 60 values in array *V* onto a new array *T* according to the following mapping table:

Position in *V*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Maps to position in *T*	-	0	1	2	47	3	4	5	6	48	7	8	9	10	49
Position in *V*	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Maps to position in *T*	11	12	13	14	50	15	16	17	51	18	19	20	21	52	22
Position in *V*	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
Maps to position in *T*	23	24	53	25	26	27	28	54	29	30	55	31	32	33	34
Position in *V*	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
Maps to position in *T*	56	35	36	37	38	57	39	40	41	42	58	43	44	45	46

NOTE 1: This re-arranges the bits, putting the data fields first and the error detection data at the end.

NOTE 2: V_{60} *is only an even parity bit (used in step 2) and is not used in the following steps.*

6. Treating each element of *T* as a binary (value *0* or *1*) coefficient, construct the polynomial:

$$\sum_{n=0}^{58} T_n x^{(58-n)}$$

7. Calculate the remainder on dividing the above polynomial by the generator polynomial:

$$x^{12} + x^{10} + x^{8} + x^{5} + x^{4} + x^{3} + 1$$

8. If the result is zero and *k* is even, the bar code was probably read correctly; if not there was at least one error.

NOTE: There can be no absolute certainty of correct reading: it is possible, though unlikely (the error pattern would have to be a multiple of the generation polynomial), that an even numbered combination of 4 or more errors will appear to be correct.

A.3 Correcting an erroneous bar code

A possible (informative) implementation of the error correction algorithm is documented below:

NOTE: It should be stressed that other implementations are possible and, depending on the reader and processing environment, may be much more efficient.

```
COPYRIGHT NOTICE: This software is provided by United States Postal Service (USPS).
It can be used only for ID-tag applications which make use of ID-tags in conformity
with UPU standard S18c. The software may be freely distributed and/or modified.  A
condition of such distribution is that this copyright notice is always included in
the comments heading the software. This software is licensed "as is" without any
other warranty of any kind. USPS expressly disclaims all warranties or conditions
including, but not limited to, implied warranties or conditions of merchantability
and fitness for particular purpose and those arising by statute or otherwise in law
or from course of dealing or usage of trade. In no event shall USPS be liable for
any direct, indirect, special, incidental, consequential or other damages arising
out of this software even if USPS has been advised of the possibility of such
damages in advance.

/*
*   This module contains the function ncorrect_to_info which validates
*   and corrects an S18c format ID-tag bar code.
*
*   Principal function in this module is:
*   ncorrect_to_info(infon *info, int *bars);
*        Correct errors in bar code
*        Convert corrected bar code into info, being:
*             cbit:         value of the Cbit
*             machine:  machine number in range 0001-3999
*             day:      day of month in range 01-31
*             time:          time of day in half hour increments, range 00-47
*             sequence: item sequence number, range 00001-24999
*
*       Returns number of errors corrected, or -1 if not correctable.
*
*    Subsidiary functions are:
*        bars_to_bits: converts bar code to bit string and ecc string
*        make_ecc:     used by correct to calculate ecc and parity bits
*        count_bits:        subfunction of correct used in parity checking
*        correct:      corrects bit string derived from bar code
*        bits_to_group:subfunction of bits_to_info
*        bits_to_info: converts corrected bit string to data values
*
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <io.h>
```

```
#include <fcntl.h>
#include <sys\stat.h>

/* National ID Tag Data */
typedef struct
    {
    unsigned cbit;      /* Cbit */
    unsigned machine;   /* machine id, range 0001-3999 */
    unsigned day;       /* day of month, range 01-31 */
    unsigned time;      /* 00-47, half hour increments */
    unsigned sequence;  /* mailpiece sequence number, 00001-24999 */
    } infon;

/* function definitions */
int  ncorrect_to_info(infon *info, int *bars);

#define DATA_LENGTH 47   /* actual length of data */
#define CRCLEN 12        /* length of CRC */
#define GENERATOR 0x1539 /* generator polynomial x12+x10+x8+x5+x4+x3+1 */
#define GENHI     0x1000 /* the x12 term where the data goes */

/*************  DATA VARIABLES ****************/

static int bits[47];           /* data bits only */
static int eccbits;            /* ecc bits extracted from *bars */
static int parbit;             /* the parity bit */
static int this_ecc;           /* ecc bits calculated from bits[] */

/* Values[] is the inverse of codes[] (in the module for generating S18c
 * bar codes), for looking up the coded value of a group. -1 indicates
 * invalid patterns. Note that this table is based on inverted-bit group
 * values used in the data section of the bar code.
 */
/*                  bit patterns                                  values    */
static int values[]= { 0, 1, 2, 3, 4, 5, 6,-1,                /* 0x00-07 */
                       8, 9,-1, 7,-1,-1,-1,-1};               /* 0x08-0F */

/*************  FUNCTION DECLARATIONS ****************/

static void bars_to_bits(int *bars);
static void make_ecc(void);
static int count_bits(int e);
static int  correct(int *bars);
static int  bits_to_group(int start, int length);
static int  bits_to_info(infon *info);

/*************  THE FUNCTIONS ***************/

static void bars_to_bits(int *bars)
    /* re-order bars[] to bits[] */
    /* used by ncorrect_to_info */
{
int i;

    for (i=0; i<3; i++)
        bits[i] = bars[i+1];     /* CM3 */  /* offset = start bit */
    for (i=3; i<7; i++)
        bits[i] = bars[i+2];     /*  M2 */  /* offset = start + E11 */
    for (i=7; i<11; i++)
        bits[i] = bars[i+3];     /*  M1 */
    for (i=11; i<15; i++)
```

```
        bits[i] = bars[i+4];     /*  M0 */
    for (i=15; i<18; i++)
        bits[i] = bars[i+5];     /*  D1 */
    for (i=18; i<22; i++)
        bits[i] = bars[i+6];     /*  D0 */
    for (i=22; i<25; i++)
        bits[i] = bars[i+7];     /*  T1 */
    for (i=25; i<29; i++)
        bits[i] = bars[i+8];     /*  T0 */
    for (i=29; i<31; i++)
        bits[i] = bars[i+9];     /*  S4 */
    for (i=31; i<35; i++)
        bits[i] = bars[i+10];    /*  S3 */
    for (i=35; i<39; i++)
        bits[i] = bars[i+11];    /*  S2 */
    for (i=39; i<43; i++)
        bits[i] = bars[i+12];    /*  S1 */
    for (i=43; i<47; i++)
        bits[i] = bars[i+13];    /*  S0 */  /* offset = start + E11..E0 */
    eccbits  = bars[4] << 11;    /* E11 */
    eccbits |= bars[9] << 10;    /* E10 */
    eccbits |= bars[14] << 9;    /* E9 */
    eccbits |= bars[19] << 8;    /* E8  */
    eccbits |= bars[23] << 7;    /* E7  */
    eccbits |= bars[28] << 6;    /* E6  */
    eccbits |= bars[32] << 5;    /* E5  */
    eccbits |= bars[37] << 4;    /* E4  */
    eccbits |= bars[40] << 3;    /* E3  */
    eccbits |= bars[45] << 2;    /* E2  */
    eccbits |= bars[50] << 1;    /* E1  */
    eccbits |= bars[55];         /* E0  */
    parbit   = bars[60];         /* Ep  */
}

static void make_ecc(void)
    /* Generate correct this_ecc and parbit matching data in bits[0..46] */
    /* Used by both ninfo_to_bars and correct */
{
int i,j;

    this_ecc = 0;
    parbit = 0;
    for (i=0; i<47; i++)
        {
         this_ecc <<= 1;             /* left-shift one bit */
         if (bits[i])
             {
             parbit ^= 1;         /* XOR the parity bit */
              this_ecc ^= GENHI;  /* XOR the data bit */
             }
         if (this_ecc & GENHI)       /* replace 12-bit carry+data with generator */
             this_ecc ^= GENERATOR;  /* binary 010100111001, & turn off GENHI */
         }

    j = this_ecc;
    for (i=0; i<12; i++)
        {
        parbit ^= (j&1);
        j >>= 1;
        }
}
```

```
static int count_bits(int e)
    /* Count and return number of bits in parameter */
    /* Used by correct */
{
int i,j;

    for (i=0,j=0; i<CRCLEN; i++)
        {
        if (e & 0x0001)
            j++;
        e >>= 1;
        }
    return(j);
}

static int correct(int *bars)
    /* This function returns the number of errors corrected as a positive
     * number if it thinks it found the correct data (0 if no errors),
     * else returns a negative error code.
     *
     * Used by ncorrect_to_info
     *
     * Input data is in global variables:
     * bits[] contains the info to be corrected.
     * eccbits contains the ecc extracted from input bars[].
     * bars[60] contains the input parity bit.
     *
     * Note this function ignores start/stop bits at bars[0,61]; uncertainty
     * as to position should be considered an additional error.
     *
     * Also, although sometimes technically possible, three or more errors
     * are considered too unreliable and are rejected as uncorrectable, so
     * the maximum value returned by this function is 2.
     */
{
int i,j;
unsigned u;
unsigned syndrome;
unsigned g; /* generate GENnn table here */
unsigned d; /* difference between syndrome and g */

        make_ecc();   /* calculate this_ecc and parbit from bits[] */
        syndrome = eccbits ^ this_ecc;       /* error syndrome */
        i = count_bits(syndrome);
        if (i==0)
            {
            /* No CRC corrections */
            if (parbit == bars[60])
                return(0);                   /* no errors detected */
            else
                return(1);                   /* parity bit error */
            }
        if (i==1)
            {
            /* 1-bit CRC error */
            if (parbit == bars[60])
                return(1);                   /* one CRC bit error */
            else
                return(2);                   /* one CRC and parity bit */
            }
```

```
if (i==2)
    {
    /* 2-bit CRC error */
    if (parbit == bars[60])
        return(2);                  /* two CRC bit errors */
    else
        return(-1);                 /* uncorrectable error */
    }

/* Now we have to look for data-bit errors; this will
 * take a while since we insist on doing it without any
 * table lookup, just to prove it's possible!
 */
g = GENHI;
g >>= 1;   /* genhi after first left shift */
for (i=0; i<DATA_LENGTH; i++)       /* for each GENxx table entry g */
    {
    g <<= 1;                        /* calculate g = this bit's ECC */
    if (g & GENHI)
        g ^= GENERATOR;

    /* syndrome matches this entry? */
    d = g ^ syndrome;  /* d = subtract this bit's ECC from syndrome */
    j = count_bits(d);
    if (j==0)
        {
        /* 1-bit data error found */
        bits[DATA_LENGTH-i-1] ^= 1;  /* correct the data error */
        make_ecc();                  /* make new parbit */
        if (parbit == bars[60])
            return(1);               /* one data bit corrected */
        else
            return(2);               /* one data and parity bit */
        }
    else if (j==1)
        {
        /* 1 data and 1 ECC error */
        bits[DATA_LENGTH-i-1] ^= 1;  /* correct the data error */
        make_ecc();                  /* make new parbit */
        if (parbit == bars[60])
            return(2);               /* one data and one ecc bit */
        else
            return(-2);              /* uncorrectable error */
        }
    else
        {
        /* Look for 2-data-bit error;
         * found if syndrome_difference d matches GENnn entry.
         * We will have to generate the whole table again to check.
         */
        u = g;             /* can't be a bit we already checked */
        for (j=i+1; j<DATA_LENGTH; j++)
            {
            u <<= 1;                 /* calculate u = 2nd bit's ECC */
            if (u & GENHI)
                u ^= GENERATOR;
            if (u == d)            /* if new ECC matches difference */
                {
                /* found 2-data-bit error */
                bits[DATA_LENGTH-i-1] ^= 1;  /* correct data errors */
                bits[DATA_LENGTH-j-1] ^= 1;
```

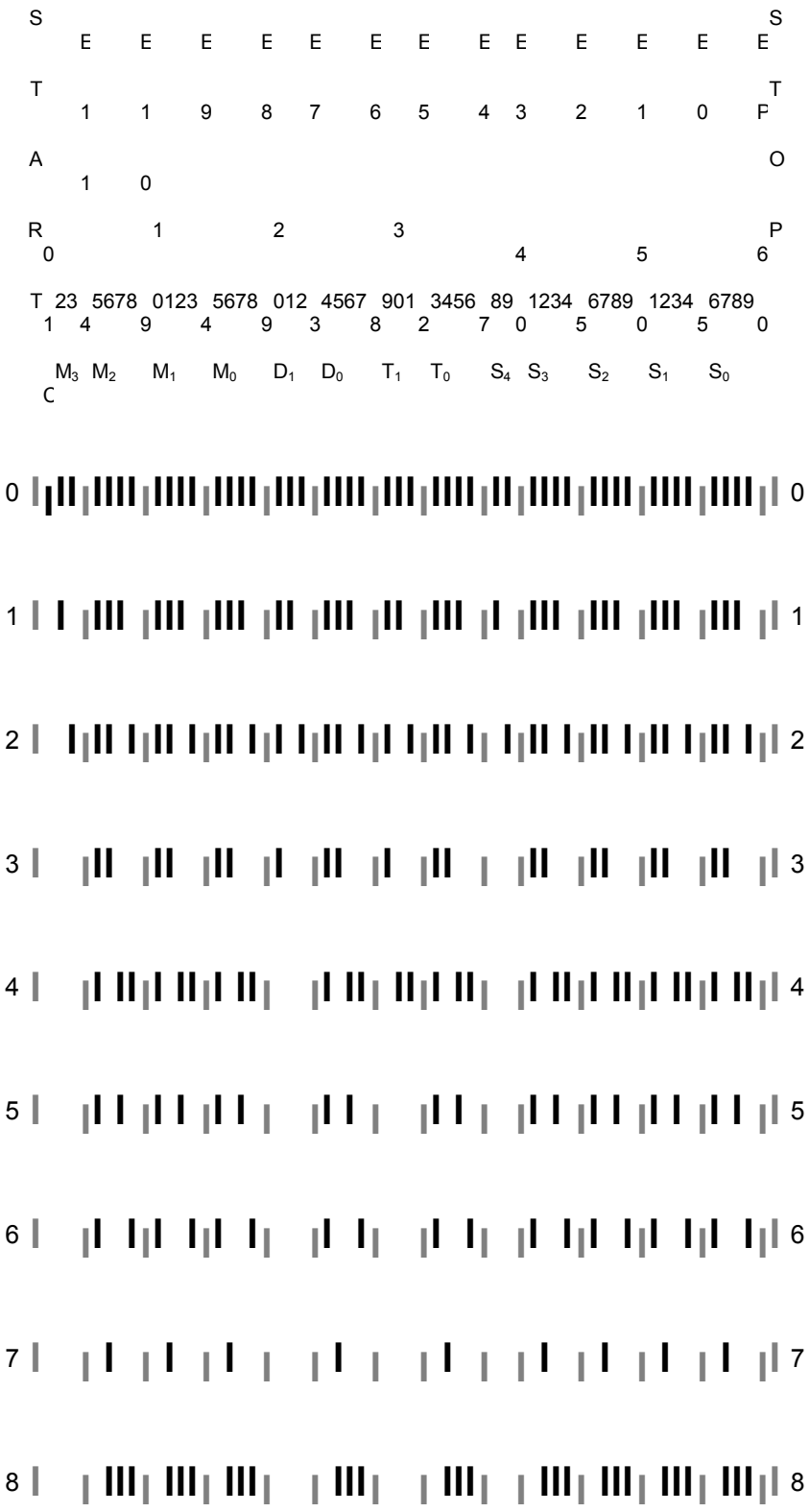
```
                        make_ecc();                 /* make new parbit */
                        if (parbit == bars[60])
                            return(2);              /* two data bits corrected */
                        else
                            return(-3);             /* uncorrectable error */
                        }
                    }
                }
            }

        return(-4);             /* couldn't correct the error */
}

static int bits_to_group(int start, int length)
    /* returns data value matching a string of bits[] */
    /* used by bits_to_inf0 */
{
int i;

    for (i=0; length; length--)
        {
        i <<= 1;
        i += bits[start++]^0x01;
        }
    return(values[i]);
}

static int bits_to_info(infon *info)
    /* Convert bits[] to info */
    /* Check info values, return error code if out-of-range */
    /* Used by ncorrect_to_info */
{
int i;
unsigned u;

    /* Cbit; */
    if (bits[0])                    /* nobar */
        info->cbit = 0;
    else
        info->cbit = 1;

    /* machine;    machine id, range 0001-3999 */
    u = bits_to_group(1,2) * 1000;  /* M3 */
    i = bits_to_group(3,4);         /* M2 */
    if (i<0)
        return(1);
    else
        u += i * 100;
    i = bits_to_group(7,4);         /* M1 */
    if (i<0)
        return(2);
    else
        u += i * 10;
    i = bits_to_group(11,4);        /* M0 */
    if (i<0)
        return(3);
    else
        u += i;
    if (u>3999)
        return(4);
    info->machine = u;
```

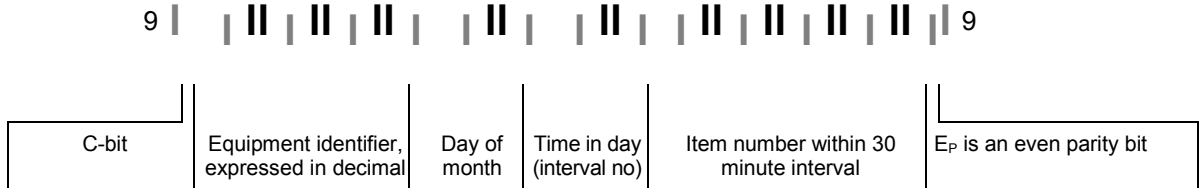
```
/* day;        day of month, range 01-31 */
i = bits_to_group(15,3);         /* D1 */
if (i<0)
    return(5);
else
    u = i * 10;
i = bits_to_group(18,4);         /* D0 */
if (i<0)
    return(6);
else
    u += i;
if (u>31)
    return(7);
info->day = u;

/* time;       00-47, half hour increments */
i = bits_to_group(22,3);         /* T1 */
if (i<0)
    return(5);
else
    u = i * 10;
i = bits_to_group(25,4);         /* T0 */
if (i<0)
    return(8);
else
    u += i;
if (u>47)
    return(9);
info->time = u;

/* sequence;   mailpiece sequence number, 00001-24999 */
i = bits_to_group(29,2);         /* S4 */
if (i>2)
    return(10);
else
    u = i * 10000;
i = bits_to_group(31,4);         /* S3 */
if (i<0)
    return(11);
else
    u += i * 1000;
i = bits_to_group(35,4);         /* S2 */
if (i<0)
    return(12);
else
    u += i * 100;
i = bits_to_group(39,4);         /* S1 */
if (i<0)
    return(13);
else
    u += i * 10;
i = bits_to_group(43,4);         /* S0 */
if (i<0)
    return(14);
else
    u += i;
if (u>24999)
    return(15);
info->sequence = u;
```

```
    return(0);                                  /* done, no errors detected */
}

int  ncorrect_to_info(infon *info, int *bars)
    /* Extract corrected info from bars */
    /* Return -1 (fail) or number of errors corrected */
{
int i;

    bars_to_bits(bars); /* extract data values including eccbits (not parbit) */

    i = correct(bars);            /* perform error correction on bits[] */
    if (i<0)
        return(-1);               /* uncorrectable error */
    else if (bits_to_info(info))  /* extract info from bits */
        return(-1);               /* out-of-range after correction */

    return(i);     /* success! - return # of errors corrected */
}
```

Annex B
(informative)

UPU ID-tag 62-position BNB bar code template





1. Format identifier is always *18B*
2. Issuer code is derived from the equipment identifier and the C-bit.
3. Calendar month is not represented in the bar code, but is derived from the comparison of day of month in the bar code with the current date.
4. Day of month is divided into tens (D_1) and units (D_0) ; value 3 can be used in the tens part only with values 0 and 1 for units.
5. Time interval (T_1 and T_0) is expressed in half hour units, with 00:00 = *00*; 00:30 = *01*; …; 23:30 = *47*.
6. Time is only given to an accuracy of thirty minutes. If desired, it is permissible to use part of the item number to express minutes within the half hour.

Bibliography

This annex provides full reference and sourcing information for all standards and other reference sources which are quoted in the above text. For references which mention specific version numbers or dates, subsequent amendments to, or revisions of, any of these publications may not be relevant. However, users of this standard are encouraged to investigate the existence and applicability of more recent editions. For references without date or version number, the latest edition of the document referred to applies.

It should be stressed that only referenced documents are listed here.

UPU standards

The UPU standards listed below are available on subscription from the UPU International Bureau:

Weltpoststrasse 4, Case postale, 3000 Berne 15, Switzerland; Tel: +41.31.350.3111; Fax: +41.31.350.3110; www.upu.int

[1] General information on UPU standards, accessible on URL http://www.upu.int

[2] Letter Post Manual

NOTE: The version applicable as at the date of publication of this standard was update 1 of September 2001. Later versions may also be used, but may be subject to change in the numbers of individual articles.