Royal Mail

# Royal Mail Mailmark® barcode L encoding and decoding

Release 1b (updated for Mailmark™ to ® branding), effective from 01/09/2015

## Disclaimer

“Whilst every effort has been made to ensure that the guidelines contained in the document are correct, Royal Mail and any other party involved in the creation of the document HEREBY STATE that the document is provided without warranty, either expressed or implied, of accuracy or fitness for purpose, AND HEREBY DISCLAIM any liability, direct or indirect, for damages or loss relating to the use of the document. The document may be modified, subject to developments in technology, changes to the standards, or new legal requirements.”



# Table of Contents

# List of Tables

# 1. Introduction

Royal Mail's Mailmark Programme defines a series of barcodes and associated data.

| Total Population | Mailmark barcodes | | | | | |
|---|---|---|---|---|---|---|
| Mailmark®barcode family | 4-state Mailmark barcodes | | | 2D data matrix Mailmark barcodes | | |
| Specific Mailmark | Mailmark 4-state barcode S | Mailmark 4-state barcode C | Mailmark 4-state barcode L | 2D Type 7 Mailmark barcode | 2D Type 9 Mailmark barcode | 2D Type 29 Mailmark barcode |

This document describes the method of encoding data into a Mailmark barcode L and the method of decoding a Mailmark barcode L to its constituent data. Mailmark Barcode L is a four-state barcode with 78 bars.

# 1.1 Conventions Used in This Document

All characters and character strings are limited to printable ASCII characters.

Accepted character values are specified within square brackets, for example, [0123456789] for decimal digits. Ranges may also be specified, for example, [0-9] is equivalent to [0123456789]. Similarly, [0-9A-F] is equivalent to [0123456789ABCDEF].

Numbers are expressed as base-ten, unless followed by a subscript that explicitly specifies the base.

# 2. Encoding

## 2.1 Encoding overview

The encoding process starts with a 26-character Application String and results in a 78-character string that represents 78 bars. The encoding process is composed of eight steps:

1. Conversion from Application String to External User Fields
2. Conversion from External User Fields to Internal User Fields
3. Conversion from Internal User Fields to Consolidated Data Value
4. Conversion from Consolidated Data Value to Data Numbers
5. Generation of Reed-Solomon Check Numbers
6. Conversion from Data Numbers and Check Numbers to Data Symbols and Check Symbols
7. Conversion from Data Symbols and Check Symbols to Extender Groups
8. Conversion from Extender Groups to Bar Identifiers

# 2.2 Encoding Details

## 2.2.1 Conversion from Application String to External User Fields

The 26-character Application String is related to the six External User Fields as shown in Table 1. The characters in the Application String are numbered left-to-right, from one to 26.

| Application String: Character Range | External User Field: Name | External User Field: String Length |
|---|---|---|
| 1-1 | Format | 1 |
| 2-2 | Version ID | 1 |
| 3-3 | Class | 1 |
| 4-9 | Supply Chain ID | 6 |
| 10-17 | Item ID | 8 |
| 18-26 | Destination Post Code plus DPS | 9 |

**Table 1. Constituent External User Fields of the Application String**

Note that the order of the allowed character values is significant.

The encoding process described herein is valid only for cases in which the value of the Version ID Expernal User Field is 1.

## 2.2.2 Conversion from External User Fields to Internal User Fields

Each External User Field is converted from a character string to an integer Internal User Field. The number of values in the Internal User Field required to represent a string of $N$ characters, each of which has $A$ allowed character values is $A^N$. A conversion process is performed for each field and is executed as follows for the first five External User Fields. (The "Destination Post Code plus DPS" field is processed using a different method, which is described later.)

1. For each field, starting from the leftmost character in the array of allowed character values (see Table 2), assign sequentially to each allowed character an integer value beginning with zero.
2. Define an integer accumulator, $a$, with an initial value of zero.
3. From the External User Field string, select the leftmost character that has not yet been processed. If all characters from the External User Field string have been processed, then processing is complete, and the Internal User Field value is $a$.
4. Set $a$ to $a$ multiplied by the number of allowed characters (see Table 2).
5. Set $a$ to $a$ plus the integer value calculated in Step 1 that corresponds to the selected character from the External User Field string.
6. Go to Step 3.

| Field Name | External User Field: Allowed Character Values | External User Field: Number of Allowed Characters | External User Field: String Length | Internal User Field: Numeric Range |
|---|---|---|---|---|
| Format | [01234] | 5 | 1 | 0-4 |
| Version ID | [1234] | 4 | 1 | 0-3 |
| Class | [0123456789ABCDE] | 15 | 1 | 0-14 |
| Supply Chain ID | [0123456789] | 10 | 6 | 0-999,999 |
| Item ID | [0123456789] | 10 | 8 | 0-99,999,999 |
| Destination Post Code plus DPS | (See text.) | (See text.) | 9 | 0-207,792,000,000 |

**Table 2. External User Field Characteristics and Their Internal User Field Ranges.**

The "Destination Post Code plus DPS" field has relatively complex rules and must be encoded using a custom method. The "Destination Post Code plus DPS" field may contain a fixed string denoting international or one of six patterns denoting a domestic sorting code. A domestic sorting code consists of an outward postcode, an inward postcode, and a Delivery Point Suffix.

The international designation is "XY11 ", a nine-character string with five trailing spaces.

Each character in a domestic sorting code belongs to one of four character types, as specified in Table 3. There are six allowed character patterns, which are listed below using the character-type

abbreviations from Table 3.

- $FNFNLLNLS$
- $FFNNLLNLS$
- $FFNNNLLNL$
- $FFNFNLLNL$
- $FNNLLNLSS$
- $FNNNLLNLS$

| Character Type | Character Type Abbreviation | External User Field: Allowed Character Values | Numeric Range |
|---|---|---|---|
| Full Alphabetic | $F$ | [ABCDEFGHIJKLMNOPQRSTUVWXYZ] | 0-25 |
| Limited Alphabetic | $L$ | [ABDEFGHJLNPQRSTUWXYZ] | 0-19 |
| Numeric | $N$ | [0123456789] | 0-9 |
| Space | $S$ | [ ] | N/A |

**Table 3. Character Types for Domestic Sorting Codes.**

The process for converting the “Destination Post Code plus DPS” External User Field to its Internal User Field is described below. Note that international designator generates an Internal User Field Value of zero. Note that spaces are not encoded. Note that the six groups of steps a-e are identical.

1. For each non-space character type, starting from the leftmost character in the array of allowed character values (see Table 3), assign sequentially to each allowed character an integer value beginning with zero.
2. Define an integer accumulator, $a$, with an initial value of zero.
3. If the “Destination Post Code plus DPS” External User Field is “XY11 “, then processing is complete, and the Internal User Field value is $a$.
4. Set $a$ to one.
5. If the character pattern of the field is $FNFNLLNLS$, then perform steps a-e below.
    a. Define an integer accumulator, $b$, with an initial value of zero.
    b. From the “Destination Post Code plus DPS” External User Field, select the leftmost character that has not yet been processed. If all non-space characters from the string have been processed, then processing is complete, and the Internal User Field value is $a$ plus $b$.
    c. Set $b$ to $b$ multiplied by the number of allowed character values for the selected character and its character type (see Table 3).
    d. Set $b$ to $b$ plus the integer value calculated in Step 1 that corresponds to the selected

character from the “Destination Post Code plus DPS” External User Field string and its character type.

e. Go to Step 5b.

6. Set $a$ to $a$ plus 5,408,000,000.

7. If the character pattern of the field is $F\,F\,N\,N\,L\,L\,N\,L\,S$, then perform steps a-e below.

   a. Define an integer accumulator, $b$, with an initial value of zero.

   b. From the “Destination Post Code plus DPS” External User Field, select the leftmost character that has not yet been processed. If all non-space characters from the string have been processed, then processing is complete, and the Internal User Field value is $a$ plus $b$.

   c. Set $b$ to $b$ multiplied by the number of allowed character values for the selected character and its character type (see Table 3).

   d. Set $b$ to $b$ plus the integer value calculated in Step 1 that corresponds to the selected character from the “Destination Post Code plus DPS” External User Field string and its character type.

   e. Go to Step 7b.

8. Set $a$ to $a$ plus 5,408,000,000.

9. If the character pattern of the field is $F\,F\,N\,N\,N\,L\,L\,N\,L$, then perform steps a-e below.

   a. Define an integer accumulator, $b$, with an initial value of zero.

   b. From the “Destination Post Code plus DPS” External User Field, select the leftmost character that has not yet been processed. If all non-space characters from the string have been processed, then processing is complete, and the Internal User Field value is $a$ plus $b$.

   c. Set $b$ to $b$ multiplied by the number of allowed character values for the selected character and its character type (see Table 3).

   d. Set $b$ to $b$ plus the integer value calculated in Step 1 that corresponds to the selected character from the “Destination Post Code plus DPS” External User Field string and its character type.

   e. Go to Step 9b.

10. Set $a$ to $a$ plus 54,080,000,000.

11. If the character pattern of the field is $F\,F\,N\,F\,N\,L\,L\,N\,L$, then perform steps a-e below.

    a. Define an integer accumulator, $b$, with an initial value of zero.

    b. From the “Destination Post Code plus DPS” External User Field, select the leftmost character that has not yet been processed. If all non-space characters from the string have been processed, then processing is complete, and the Internal User Field value is $a$ plus $b$.

    c. Set $b$ to $b$ multiplied by the number of allowed character values for the selected character and its character type (see Table 3).

    d. Set $b$ to $b$ plus the integer value calculated in Step 1 that corresponds to the selected character from the “Destination Post Code plus DPS” External User Field string and its character type.

    e. Go to Step 11b.

12. Set $a$ to $a$ plus 140,608,000,000.
13. If the character pattern of the field is $F\ N\ N\ L\ L\ N\ L\ S\ S$, then perform steps a-e below.
    a. Define an integer accumulator, $b$, with an initial value of zero.
    b. From the "Destination Post Code plus DPS" External User Field, select the leftmost character that has not yet been processed. If all non-space characters from the string have been processed, then processing is complete, and the Internal User Field value is $a$ plus $b$.
    c. Set $b$ to $b$ multiplied by the number of allowed character values for the selected character and its character type (see Table 3).
    d. Set $b$ to $b$ plus the integer value calculated in Step 1 that corresponds to the selected character from the "Destination Post Code plus DPS" External User Field string and its character type.
    e. Go to Step 13b.
14. Set $a$ to $a$ plus 208,000,000.
15. If the character pattern of the field is $F\ N\ N\ N\ L\ L\ N\ L\ S$, then perform steps a-e below.
    a. Define an integer accumulator, $b$, with an initial value of zero.
    b. From the "Destination Post Code plus DPS" External User Field, select the leftmost character that has not yet been processed. If all non-space characters from the string have been processed, then processing is complete, and the Internal User Field value is $a$ plus $b$.
    c. Set $b$ to $b$ multiplied by the number of allowed character values for the selected character and its character type (see Table 3).
    d. Set $b$ to $b$ plus the integer value calculated in Step 1 that corresponds to the selected character from the "Destination Post Code plus DPS" External User Field string and its character type.
    e. Go to Step 15b.

## 2.2.3 Conversion from Internal User Fields to Consolidated Data Value

The six Internal User Fields are converted to a single 93-bit Consolidated Data Value. The “Destination Post Code plus DPS” field will occupy the most significant bits of the Consolidated Data Value, and the Version ID field be represented in the least significant bits of the Consolidated Data Value. Note that the order of processing differs from the order of the fields as they appear in Table 1 and Table 2.

The conversion process is as follows. At the conclusion of the process, the Consolidated Data Value is the value of the accumulator, $a$.

1. Define an integer accumulator, $a$, with an initial value of zero.
2. Set $a$ to $a$ plus the value of the “Destination Post Code plus DPS” Internal User Field.
3. Set $a$ to $a$ multiplied by 100,000,000.
4. Set $a$ to $a$ plus the value of the Item ID Internal User Field.
5. Set $a$ to $a$ multiplied by 1,000,000.
6. Set $a$ to $a$ plus the value of the Supply Chain ID Internal User Field.
7. Set $a$ to $a$ multiplied by 15.
8. Set $a$ to $a$ plus the value of the Class Internal User Field.
9. Set $a$ to $a$ multiplied by 5.
10. Set $a$ to $a$ plus the value of the Format Internal User Field.
11. Set $a$ to $a$ multiplied by 4.
12. Set $a$ to $a$ plus the value of the Version ID Internal User Field.

The Consolidated Data Value will be in the range 0-6,233,760,000,029,999,999,999,999,999 ($0_{16}$-1424 7206 BD41 FDF9 3342 $FFFF_{16}$).

## 2.2.4 Conversion from Consolidated Data Value to Data Numbers

The 93-bit Consolidated Data Value is converted into 19 numbers that are suitable for use with a Galois Field of 32 values. Normally, each of the 19 numbers would range from zero to 31. However, because of issues regarding synchronisation and orientation, 11 of the numbers have 30 values and eight have 32 values.

The 19 numbers will be referred to as $D_0$ through $D_{18}$. D stands for "data". $D_0$ will be considered the most significant and $D_{18}$ the least significant. Note the contrast between this ordering and bit significance (where the least significant bit is bit 0). Table 4 lists the Data Numbers from most significant to least significant and how many values each number may take.

| Data Number | Number of Values |
|---|---|
| $D_0$ | 30 |
| $D_1$ | 30 |
| $D_2$ | 30 |
| $D_3$ | 30 |
| $D_4$ | 30 |
| $D_5$ | 30 |
| $D_6$ | 30 |
| $D_7$ | 30 |
| $D_8$ | 30 |
| $D_9$ | 30 |
| $D_{10}$ | 30 |
| $D_{11}$ | 32 |
| $D_{12}$ | 32 |
| $D_{13}$ | 32 |
| $D_{14}$ | 32 |
| $D_{15}$ | 32 |
| $D_{16}$ | 32 |
| $D_{17}$ | 32 |
| $D_{18}$ | 32 |

**Table 4. Data Numbers.**

The process of calculating the Data Numbers is as follows. The division operations are integer operations, each of which results in an integer quotient and an integer remainder.

1. Define the variable, $x$, with an initial value equal to the Consolidated Data Value.
2. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{18}$.
3. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{17}$.
4. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{16}$.

5. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{15}$.
6. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{14}$.
7. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{13}$.
8. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{12}$.
9. Divide $x$ by 32, save the quotient in $x$, and save the remainder in $D_{11}$.
10. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_{10}$.
11. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_9$.
12. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_8$.
13. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_7$.
14. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_6$.
15. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_5$.
16. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_4$.
17. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_3$.
18. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_2$.
19. Divide $x$ by 30, save the quotient in $x$, and save the remainder in $D_1$.
20. Divide $x$ by 30, and save the quotient in $D_0$.

## 2.2.5 Generation of Reed-Solomon Check Numbers

Seven Reed-Solomon codes (Check Numbers) are calculated from the 19 Data Numbers. The coding uses a Galois field with 32 values. Each Data Number is considered to be five bits, even those that have only 30 values.

Given the primitive polynomial: $p(x) = x^5 + x^2 + 1,$

and given the generator polynomial: $G = x^7 + 5x^6 + 9x^5 + 5x^4 + 26x^3 + 17x^2 + 25x + 22,$

Construct the polynomial Numbers $P = \sum_{n=0}^{18} D_n X^{25-n}$ where $D_n$ are the Data Numbers

Divide P by G and save the remainder R = $\sum_{n=0}^{6}$ $C_n x^{(6-n)}$ , where $C_n$ are the Check Numbers

## 2.2.6 Conversion from Data Numbers and Check Numbers to Data Symbols and Check Symbols

Table 5 shows the conversion from Data Numbers ($D_0$-$D_{18}$) and Check Numbers ($C_0$-$C_6$) to Data/Check Symbols. Data Symbols and Check Symbols are six-bit numbers.

Note that some Data Numbers are limited to the range 0-29.

Note that Table 0 symbols contain an odd number of binary 1s, while Table 1 symbols contain a non-zero, even number of binary 1s.

| Data/Check Number | Data/Check Symbol (Table 0) Use for: $D_{11}$, $D_{12}$, $D_{13}$, $D_{14}$, $D_{15}$, $D_{16}$, $D_{17}$, $D_{18}$,$C_0$, $C_1$, $C_2$, $C_3$, $C_4$, $C_5$, $C_6$ | ck Symbol 1) for: , $D_4$, $D_5$, $D_6$, $D_7$, $D_9$, $D_{10}$ |
|---|---|---|
| 0 | $000001_2$ | $00011_2$ |
| 1 | $000010_2$ | $000101_2$ |
| 2 | $000100_2$ | $000110_2$ |
| 3 | $000111_2$ | $001001_2$ |
| 4 | $001000_2$ | $001010_2$ |
| 5 | $001011_2$ | $001100_2$ |
| 6 | $001101_2$ | $001111_2$ |
| 7 | $001110_2$ | $010001_2$ |
| 8 | $010000_2$ | $010010_2$ |
| 9 | $010011_2$ | $010100_2$ |
| 10 | $010101_2$ | $010111_2$ |
| 11 | $010110_2$ | $011000_2$ |
| 12 | $011001_2$ | $011011_2$ |
| 13 | $011010_2$ | $011101_2$ |
| 14 | $011100_2$ | $011110_2$ |
| 15 | $011111_2$ | $100001_2$ |
| 16 | $100000_2$ | $100010_2$ |
| 17 | $100011_2$ | $100100_2$ |
| 18 | $100101_2$ | $100111_2$ |
| 19 | $100110_2$ | $101000_2$ |
| 20 | $101001_2$ | $101011_2$ |
| 21 | $101010_2$ | $101101_2$ |
| 22 | $101100_2$ | $101110_2$ |
| 23 | $101111_2$ | $110000_2$ |
| 24 | $110001_2$ | $110011_2$ |
| 25 | $110010_2$ | $110101_2$ |
| 26 | $110100_2$ | $110110_2$ |
| 27 | $110111_2$ | $111001_2$ |
| 28 | $111000_2$ | $111010_2$ |
| 29 | $111011_2$ | $111100_2$ |
| 30 | $111101_2$ | |
| 31 | $111110_2$ | - |

**Table 5. Relationship Between Data/Check Numbers and Data/Check Symbols.**

## 2.2.7 Conversion from Data Symbols and Check Symbols to Extender Groups

The 19 Data Symbols and seven Check Symbols are logical symbols. Their sequence is related to the mathematical processes that formed them. The Data and Check Symbols are reordered into physical symbols called Extender Groups. The resulting sequence of Extender Groups is designed to reduce the probability of incorrect decoding.

| Data/Check Symbol (Logical) | Extender Group (Physical) |
|---|---|
| $D_0$ | $E_2$ |
| $D_1$ | $E_5$ |
| $D_2$ | $E_7$ |
| $D_3$ | $E_8$ |
| $D_4$ | $E_{13}$ |
| $D_5$ | $E_{14}$ |
| $D_6$ | $E_{15}$ |
| $D_7$ | $E_{16}$ |
| $D_8$ | $E_{21}$ |
| $D_9$ | $E_{22}$ |
| $D_{10}$ | $E_{23}$ |
| $D_{11}$ | $E_0$ |
| $D_{12}$ | $E_1$ |
| $D_{13}$ | $E_3$ |
| $D_{14}$ | $E_4$ |
| $D_{15}$ | $E_6$ |
| $D_{16}$ | $E_9$ |
| $D_{17}$ | $E_{10}$ |
| $D_{18}$ | $E_{11}$ |
| $C_0$ | $E_{12}$ |
| $C_1$ | $E_{17}$ |
| $C_2$ | $E_{18}$ |
| $C_3$ | $E_{19}$ |
| $C_4$ | $E_{20}$ |
| $C_5$ | $E_{24}$ |
| $C_6$ | $E_{25}$ |

**Table 6. Relationship Between Data/Check Symbols and Physical Extender Groups.**

# 2.2.8 Conversion from Extender Groups to Bar Identifiers

The 26 Extender Groups are assigned to groups of three consecutive bars within the 78-bar code. Each of the six bits within the Extender Group is mapped to one of the three ascenders or one of the three descenders in the same three-bar group. The mapping is shown in Table 7. Bar 1 is the leftmost bar, and Bar 78 is the rightmost bar. Each six-bit Extender Group is broken up into the most significant three bits (high, $E_{nH}$) and the least significant three bits (low, $E_{nL}$). The most significant bit corresponds to the leftmost bar.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Ascender:** | | | | | | | | |
| **Bar: Descender:** | | | | | | | | |
| **Ascender:** | **E9L** | **E10H** | **E11L** | **E12H** | **E13L** | **E14H** | **E15L** | **E16H** | **E17L** |
| **Bar: Descender:** | **28-30** | **31-33** | **34-36** | **37-39** | **40-42** | **43-45** | **46-48** | **49-51** | **52-54** |
| | **E9 H** | **E10 L** | **E1 1 H** | **E12L** | **E13H** | **E14L** | **E15H** | **E16L** | **E7H** |
| **Ascender:** | **E18H** | **E19L** | **E20H** | **E21L** | **E22H** | **E23L** | **E24H** | **E25L** |
| **Bar: Descender:** | **55-57** | **58-60** | **61-63** | **64-66** | **67-69** | **70-72** | **73-75** | **76-78** |
| | **E18L** | **E1 9 H** | **E2 0 L** | **E2 1 H** | **E22 L** | **E23H** | **E24L** | **E25H** |

**Table 7. Mapping of Extender Groups to Bar Ascenders and Descenders.**

The ascender and descender status of each of the 78 bars is then used to identify each bar with a single character, “A”, “D”, “F”, or “T”, that identifies the entire bar. The Bar Identifiers are specified in Table 8.

| Bar Identifier | Bar Description | Ascender Present? | Descender Present? |
|---|---|---|---|
| A | Ascender | Yes | No |
| D | Descender | No | Yes |
| F | Full | Yes | Yes |
| T | Tracker | No | No |

**Table 8. Bar Descriptions.**

The Bar Identifiers are then concatenated, such that the first character represents Bar 1 and the last character represents Bar 78. The resulting 78-character string is the final output of the encoding process.

# 2.3 Encoding Examples

This section contains Mailmark barcode L encoding examples. Data values are shown for significant stages in the decoding process. The first example involves simple input data for which the Internal User Fields are close to or at their minimum values. The second example involves more complex input data.

## 2.3.1 Example 1

Application String: “11000000000000000XY11 ”
Note that the Application String has five space characters at the end.

| Field Name | External User Field | Internal User Field |
|---|---|---|
| Format | “1” | 1 |
| Version ID | “1” | 0 |
| Class | “0” | 0 |
| Supply Chain ID | “000000” | 0 |
| Item ID | “0000000” | 0 |
| Destination Post Code plus DPS | “XY11 “ | 0 |

**Table 9. External User Fields and Internal User Fields, Example 1.**

Note that the “Destination Post Code plus DPS” External User Field has five space characters at the end.

Consolidated Data Value: 4

| Data/Check Name | Data Number | Check Number | Symbol | Extender Group Name |
|---|---|---|---|---|
| $D_0$ | 0 | | 3 | $E_2$ |
| $D_1$ | 0 | | 3 | $E_5$ |
| $D_2$ | 0 | | 3 | $E_7$ |
| $D_3$ | 0 | | 3 | $E_8$ |
| $D_4$ | 0 | | 3 | $E_{13}$ |
| $D_5$ | 0 | | 3 | $E_{14}$ |
| $D_6$ | 0 | | 3 | $E_{15}$ |
| $D_7$ | 0 | | 3 | $E_{16}$ |
| $D_8$ | 0 | | 3 | $E_{21}$ |
| $D_9$ | 0 | | 3 | $E_{22}$ |
| $D_{10}$ | 0 | | 3 | $E_{23}$ |
| $D_{11}$ | 0 | | 1 | $E_0$ |
| $D_{12}$ | 0 | | 1 | $E_1$ |
| $D_{13}$ | 0 | | 1 | $E_3$ |
| $D_{14}$ | 0 | | 1 | $E_4$ |
| $D_{15}$ | 0 | | 1 | $E_6$ |
| $D_{16}$ | 0 | | 1 | $E_9$ |
| $D_{17}$ | 0 | | 1 | $E_{10}$ |
| $D_{18}$ | 4 | | 8 | $E_{11}$ |
| $C_0$ | | 20 | 41 | $E_{12}$ |
| $C_1$ | | 1 | 2 | $E_{17}$ |
| $C_2$ | | 20 | 41 | $E_{18}$ |
| $C_3$ | | 7 | 14 | $E_{19}$ |
| $C_4$ | | 14 | 28 | $E_{20}$ |
| $C_5$ | | 11 | 22 | $E_{24}$ |
| $C_6$ | | 18 | 37 | $E_{25}$ |

**Table 10. Data Numbers, Check Numbers, and Data/Check Symbols, Example 1.**

| Extender Group Name | Extender Group High Bits | Extender Group Low Bits | Bar Identifiers |
|---|---|---|---|
| $E_0$ | $000_2$ | $001_2$ | TTD |
| $E_1$ | $000_2$ | $001_2$ | TTA |
| $E_2$ | $000_2$ | $011_2$ | TDD |
| $E_3$ | $000_2$ | $001_2$ | TTA |
| $E_4$ | $000_2$ | $001_2$ | TTD |
| $E_5$ | $000_2$ | $011_2$ | TAA |
| $E_6$ | $000_2$ | $001_2$ | TTD |
| $E_7$ | $000_2$ | $011_2$ | TAA |
| $E_8$ | $000_2$ | $011_2$ | TDD |
| $E_9$ | $000_2$ | $001_2$ | TTA |
| $E_{10}$ | $000_2$ | 001 | TTD |
| $E_{11}$ | $001_2$ | $000_2$ | TTD |
| $E_{12}$ | $101_2$ | $001_2$ | ATF |
| $E_{13}$ | $000_2$ | $011_2$ | TAA |
| $E_{14}$ | $000_2$ | $011_2$ | TDD |
| $E_{15}$ | $000_2$ | $011_2$ | TAA |
| $E_{16}$ | $000_2$ | $011_2$ | TDD |
| $E_{17}$ | $000_2$ | $010_2$ | TAT |
| $E_{18}$ | $101_2$ | $001_2$ | ATF |
| $E_{19}$ | $001_2$ | $110_2$ | AAD |
| $E_{20}$ | 011 | $100_2$ | DAA |
| $E_{21}$ | $000_2$ | $011_2$ | TAA |
| $E_{22}$ | $000_2$ | $011_2$ | TDD |
| $E_{23}$ | $000_2$ | $011_2$ | TAA |
| $E_{24}$ | $010_2$ | $110_2$ | DFT |
| $E_{25}$ | $100_2$ | $101_2$ | FTA |

**Table 11. High/Low Bits of Extender Groups and Bar Identifiers, Example 1.**

Bar Identifiers:
“TTDTTATDDTTATTDTAATTDTAATDDTTATTDTTDATFTAATDDTAATDDTATATFAADDAATAATD
DTAADFTFTA”

## 2.3.2 Example 2

Application String: “41038422416563762EF61AH8T ”

Note that the Application String has one space character at the end.

| Field Name | External User Field | Internal User Field |
|---|---|---|
| Format | “4” | 4 |
| Version ID | “1” | 0 |
| Class | “0” | 0 |
| Supply Chain ID | “384224” | 384,224 |
| Item ID | “16563762” | 16,563,762 |
| Destination Post Code plus DPS | “EF61AH8T “ | 6,284,881,375 |

**Table 12. External User Fields and Internal User Fields, Example 2.**

Note that the “Destination Post Code plus DPS” External User Field has one space character at the end.

Consolidated Data Value: 188,546,441,254,969,128,715,267,216 (9B F643 0FA7 AF14 C247 $AC90_{16}$)

| Data/Check Name | Data Number | Check Number | Symbol | Extender Group Name |
|---|---|---|---|---|
| $D_0$ | 0 | | 3 | $E_2$ |
| $D_1$ | 8 | | 18 | $E_5$ |
| $D_2$ | 21 | | 45 | $E_7$ |
| $D_3$ | 10 | | 23 | $E_8$ |
| $D_4$ | 29 | | 60 | $E_{13}$ |
| $D_5$ | 1 | | 5 | $E_{14}$ |
| $D_6$ | 29 | | 60 | $E_{15}$ |
| $D_7$ | 21 | | 45 | $E_{16}$ |
| $D_8$ | 2 | | 6 | $E_{21}$ |
| $D_9$ | 24 | | 51 | $E_{22}$ |
| $D_{10}$ | 15 | | 33 | $E_{23}$ |
| $D_{11}$ | 2 | | 4 | $E_0$ |
| $D_{12}$ | 19 | | 38 | $E_1$ |
| $D_{13}$ | 1 | | 2 | $E_3$ |
| $D_{14}$ | 4 | | 8 | $E_4$ |
| $D_{15}$ | 15 | | 31 | $E_6$ |
| $D_{16}$ | 11 | | 22 | $E_9$ |
| $D_{17}$ | 4 | | 8 | $E_{10}$ |
| $D_{18}$ | 16 | | 32 | $E_{11}$ |
| $C_0$ | | 19 | 38 | $E_{12}$ |
| $C_1$ | | 7 | 14 | $E_{17}$ |
| $C_2$ | | 9 | 19 | $E_{18}$ |
| $C_3$ | | 8 | 16 | $E_{19}$ |
| $C_4$ | | 6 | 13 | $E_{20}$ |
| $C_5$ | | 16 | 32 | $E_{24}$ |
| $C_6$ | | 16 | 32 | $E_{25}$ |

**Table 13. Data Numbers, Check Numbers, and Data/Check Symbols, Example 2.**

| Extender Group Name | Extender Group High Bits | Extender | Bar Identifiers |
|---|---|---|---|
| $E_0$ | $000_2$ | | DTT |
| $E_1$ | $100_2$ | $110_2$ | FAT |
| $E_2$ | $000_2$ | $011_2$ | TDD |
| $E_3$ | $000_2$ | $010_2$ | TAT |
| $E_4$ | $001_2$ | $000_2$ | TTA |
| $E_5$ | $010_2$ | $010_2$ | TFT |
| $E_6$ | $011_2$ | $111_2$ | DFF |
| $E_7$ | $101_2$ | $101_2$ | FTF |
| $E_8$ | $010_2$ | $111_2$ | DFD |
| $E_9$ | $010_2$ | $110_2$ | AFT |
| $E_{10}$ | $001_2$ | $000_2$ | TTA |
| $E_{11}$ | $100_2$ | $000_2$ | DTT |
| $E_{12}$ | $100_2$ | $110_2$ | FDT |
| $E_{13}$ | $111_2$ | $100_2$ | FDD |
| $E_{14}$ | $000_2$ | $101_2$ | DTD |
| $E_{15}$ | $111_2$ | $100_2$ | FDD |
| $E_{16}$ | $101_2$ | $101_2$ | FTF |
| $E_{17}$ | $001_2$ | $110_2$ | AAD |
| $E_{18}$ | $010_2$ | $011_2$ | TFD |
| $E_{19}$ | $010_2$ | $000_2$ | TDT |
| $E_{20}$ | $001_2$ | $101_2$ | DTF |
| $E_{21}$ | $000_2$ | $110_2$ | AAT |
| $E_{22}$ | $110_2$ | $011_2$ | AFD |
| $E_{23}$ | $100_2$ | $001_2$ | DTA |
| $E_{24}$ | $100_2$ | $000_2$ | ATT |
| $E_{25}$ | $100_2$ | $000_2$ | DTT |

**Table 14. High/Low Bits of Extender Groups and Bar Identifiers, Example 2.**

Bar Identifiers:
“DTTFATTDDTATTTATFTDFFFTFDFDAFTTTADTTFDTFDDDTDFDDFTFAADTFDTDTDTFAATAF
DDTAATTDTT”

# 3. Decoding

## 3.1 Decoding Considerations

Barcode encoding is unambiguous, in that the input Application String is always assumed to be free of errors. Decoding, however, involves ambiguous input information. The representation of bars as Bar Identifiers may be corrupted. For example, the bar code may be upside down. Bars may be missing or obscured. Noise may interfere with the intended imaging and interpretation of a bar. The corruption of bar data may lead to confusion with other four-state bar codes.

The Mailmark barcode L is designed to allow correct decoding from incomplete and/or corrupted bar data. The design also contains features to minimise the potential of one type of Mailmark four-state barcode being decoded as another type.

Reed-Solomon error correction is a powerful technique that is used for the Mailmark barcode L. However, every error-correction technique, including Reed-Solomon, has limits. The Reed-Solomon decoding process indicates the amount of error correction used. The more error correction is accepted, the more incorrect reads will occur. A prudent decoding implementation will not necessarily accept a decode that uses all of the error correction available. The specifics of a decoder implementation should take into account the quality of its input as well as the environment in which it functions, specifically the other similar bar codes and image patterns it may encounter.

### 3.1.1 Rotation

A prudent decoding implementation should attempt to decode both right-side-up (left-to-right) and upside-down (right-to-left) interpretations. The Mailmark barcode L is designed such that the incorrect rotation interpretation of an otherwise correct bar pattern will not be decoded. (The amount of error correction that would be required always exceeds the amount of error correction available.)

### 3.1.2 Shift

"Shift" refers to corruption of the bars at the ends of the barcode. For example, the two left-most bars may be obscured and absent from the input Bar Identifiers, or noise may add a phantom bar to the right end of the bar code. In these cases, the number of bars in the input may not equal the number expected for a Mailmark barcode L. A high-performance decoding implementation should attempt to decode multiple permutations of bar arrangements in the context of the correct number of bars.

A bar-code design that is based on three-bar groups (like the Mailmark barcode L) can have a relatively high susceptibility to decode errors for shifts of three bars. The Mailmark barcode L is designed such

that a three-bar shift of an otherwise correct bar pattern will not be decoded. (The amount of error correction that would be required always exceeds the amount of error correction available.)

## 3.1.3 Confusion Between Mailmark barcode types

The Mailmark barcode family is designed such that the decoding of any barcode type as a different barcode type is possible only with the maximum amount of Reed-Solomon error correction available for the barcode type that decoded. For example, a Mailmark barcode L with 78 bars could have the right-most 13 bars obscured, leaving a 66-bar code for which a Mailmark barcode C decode would be attempted. The decoding of the otherwise correct Mailmark barcode L could be decoded as an Mailmark barcode C, but only with the maximum amount of correction available for Mailmark barcode C (6). A prudent decoding implementation will take this into account.

# 3.2 Decoding Overview

The input to the decoding process is a character string that represents bars. The output of the process, if successful, is a 26-character Application String. The decoding process is composed of the following steps:

1. Conversion from Bar Identifiers to Extender Groups
2. Conversion from Extender Groups to Data Symbols and Check Symbols
3. Conversion from Data Symbols and Check Symbols to Data Numbers and Check Numbers
4. Reed-Solomon Decoding
5. Conversion from Data Numbers to Consolidated Data Value
6. Conversion from Consolidated Data Value to Internal User Fields
7. Conversion from Internal User Fields to External User Fields
8. Conversion from External User Fields to Application String

Note that certain tables from Section 2, “Encoding”, are referenced from within this section.

# 3.3 Decoding Details

## 3.3.1 Conversion from Bar Identifiers to Extender Groups

Bars are represented as a string of Bar Identifiers that are specified in Table 8. The bars are grouped and converted to high and low bits of Extender Groups, as specified in Table 7. Bar 1 is the leftmost bar, and Bar 78 is the rightmost bar. The most significant three bits of a six-bit Extender Group ($E_n$) are represented by the high bits ($E_{nH}$), and the least significant three bits are represented by the low bits ($E_{nL}$). There are 26 Extender Groups.

## 3.3.2 Conversion from Extender Groups to Data Symbols and Check Symbols

The 26 Extender Groups are related to 19 Data Symbols and seven Check Symbols as specified in Table 15. The arrangement of the symbols is designed to reduce the possibility of incorrect decoding.

| Extender Group (Physical) | Data/Check Symbol (Logical) |
|---|---|
| $E_0$ | $D_{11}$ |
| $E_1$ | $D_{12}$ |
| $E_2$ | $D_0$ |
| $E_3$ | $D_{13}$ |
| $E_4$ | $D_{14}$ |
| $E_5$ | $D_1$ |
| $E_6$ | $D_{15}$ |
| $E_7$ | $D_2$ |
| $E_8$ | $D_3$ |
| $E_9$ | $D_{16}$ |
| $E_{10}$ | $D_{17}$ |
| $E_{11}$ | $D_{18}$ |
| $E_{12}$ | $C_0$ |
| $E_{13}$ | $D_4$ |
| $E_{14}$ | $D_5$ |
| $E_{15}$ | $D_6$ |
| $E_{16}$ | $D_7$ |
| $E_{17}$ | $C_1$ |
| $E_{18}$ | $C_2$ |
| $E_{19}$ | $C_3$ |
| $E_{20}$ | $C_4$ |
| $E_{21}$ | $D_8$ |
| $E_{22}$ | $D_9$ |
| $E_{23}$ | $D_{10}$ |
| $E_{24}$ | $C_5$ |
| $E_{25}$ | $C_6$ |

**Table 15. Relationship Between Extender Groups and Data/Check Symbols**

## 3.3.3 Conversion from Data Symbols and Check Symbols to Data Numbers and Check Numbers

The logical Data and Check Symbols are converted to Data and Check Numbers as specified in Table 5. Refer to the Data/Check Symbol column (Table 0 or Table 1) appropriate for the symbol being converted, as specified in the table heading.

If the Data/Check Symbol value does not appear in the relevant column of Table 5, set the value of the corresponding Data/Check Number to -1. This value represents a known error.

## 3.3.4 Reed-Solomon Decoding

A Reed-Solomon error-correction algorithm is used to transform the 19 Data Numbers and seven Check Numbers into seven Data Numbers. The relevant Reed-Solomon error-correction characteristics are described in Section 2.2.5. Some Data Numbers may be changed during the transformation. It is possible that the Reed-Solomon decoding process may fail, in which case further decoding is impossible.

## 3.3.5 Conversion from Data Numbers to Consolidated Data Value

The Data Numbers are converted into a single Consolidated Data Value through a series of multiply/add operations according to the number of values assigned to each Data Number, as specified in Table 4. The process is described below. The final value of $x$ is the Consolidated Data Value.

1. Define the variable, $x$, with an initial value of $D_0$.
2. For each Data Number $D_n$, for values of $n$ from 1 to 10, perform the steps a-b below.
    a. Set $x$ to $x$ multiplied by 30.
    b. Set $x$ to $x$ plus $D_n$.
3. For each Data Number $D_n$, for values of $n$ from 11 to 18, perform the steps a-z below.
    a. Set $x$ to $x$ multiplied by 32.
    b. Set $x$ to $x$ plus $D_n$.

## 3.3.6 Conversion from Consolidated Data Value to Internal User Fields

The Consolidated Data Value is converted into six Internal User Fields, each of which is an integer. The conversion steps are as follows. The division operations are integer operations, each of which results in an integer quotient and an integer remainder.

1. Define the variable, $x$, with an initial value equal to the Consolidated Data Value.
2. Divide $x$ by 4, save the quotient in $x$. The Version ID Internal User Field is the remainder.
3. If the Version ID Internal User Field is not equal to zero, then processing is complete, and the decoding process fails.
4. Divide $x$ by 5, save the quotient in $x$. The Format Internal User Field is the remainder.
5. Divide $x$ by 15, save the quotient in $x$. The Class Internal User Field is the remainder.
6. Divide $x$ by 1,000,000, save the quotient in $x$. The Supply Chain ID Internal User Field is the remainder.
7. Divide $x$ by 100,000,000, save the quotient in $x$. The Item ID Internal User Field is the remainder. I
8. f $x$ is less than 207,792,000,001, then the "Destination Post Code plus DPS" Internal User Field is $x$, otherwise processing is complete, and the encoding process fails.

## 3.3.7 Conversion from Internal User Fields to External User Fields

Each of the six Internal User Fields is converted from an integer to an External User Field character string. The conversion process is performed independently for each field. The process for five of the fields (all but Destination Post Code plus DPS) is executed as follows. The division operations are integer operations, each of which results in an integer quotient and an integer remainder.

1. For each field, starting from the leftmost character in the array of allowed character values (see Table 2), assign sequentially to each allowed character an integer value beginning with zero.
2. Define an integer variable, $x$, with an initial value of the Internal User Field.
3. Define a character string, $s$, with an initial value of an empty string with zero characters.
4. Repeat the following steps a-c until $x$ is equal to zero, at which point processing is complete for the field.
    a. Divide $x$ by the number of allowed characters (Table 2), and save the quotient in $x$.
    b. Use the remainder from Step 4a as an index to the list of allowed character values (Table 2).
    c. Set $s$ to the concatenation of the indexed character from Step 4b and $s$. (In other words, stick the indexed character onto the left side of the character string $s$.)

The conversion process for the "Destination Post Code plus DPS" field is unique, and is executed as follows. The division operations are integer operations, each of which results in an integer quotient and an integer remainder.

1. Define an integer variable, $x$, with an initial value of the "Destination Post Code plus DPS"

Internal User Field.

2. If $x$ is less than 1, processing is complete, and the External User Field is "XY11 ". Note that the field has five trailing space characters.
3. Set $x$ to $x$-1.
4. If $x$ is less than 5,408,000,000, then perform the steps a-d below.
    a. Define a character string, $s$, with an initial value of " " (one space character).
    b. Define a character string, $t$, with a value of "FNFNLLNL".
    c. For each character, $c$, in $t$, starting from the right-most character, perform the steps i-iv below.
        i. Divide $x$ by the number of allowed characters for character type abbreviation $c$ as specified in Table 3.
        ii. Use the remainder from Step 4ci as an index into the allowed character values for the character type abbreviation $c$. (See Table 3.)
        iii. Set $s$ to the concatenation of the indexed character from Step 4cii and $s$. (In other words, stick the indexed character onto the left side of the character string, $s$.)
        iv. Set $x$ to the quotient from Step 4ci.
    d. Processing is complete, and the External User Field is $s$.
5. Set $x$ to $x$-5,408,000,000.
6. If $x$ is less than 5,408,000,000, then perform the steps a-d below.
    a. Define a character string, $s$, with an initial value of " " (one space character).
    b. Define a character string, $t$, with a value of "FFNNLLNL".
    c. For each character, $c$, in $t$, starting from the right-most character, perform the steps i-iv below.
        i. Divide $x$ by the number of allowed characters for character type abbreviation $c$ as specified in Table 3.
        ii. Use the remainder from Step 6ci as an index into the allowed character values for the character type abbreviation $c$. (See Table 3.)
        iii. Set $s$ to the concatenation of the indexed character from Step 6cii and $s$. (In other words, stick the indexed character onto the left side of the character string, $s$.)
        iv. Set $x$ to the quotient from Step 6ci.
    d. Processing is complete, and the External User Field is $s$.
7. Set $x$ to $x$-5,408,000,000.
8. If $x$ is less than 54,080,000,000, then perform the steps a-d below.
    a. Define a character string, $s$, with an initial value of an empty string.
    b. Define a character string, $t$, with a value of "FFNNNLLNL".
    c. For each character, $c$, in $t$, starting from the right-most character, perform the steps i-iv below.
        i. Divide $x$ by the number of allowed characters for character type abbreviation $c$ as specified in Table 3.

ii. Use the remainder from Step 8ci as an index into the allowed character values for the character type abbreviation $c$. (See Table 3.)

iii. Set $s$ to the concatenation of the indexed character from Step 8cii and $s$. (In other words, stick the indexed character onto the left side of the character string, $s$.)

iv. Set $x$ to the quotient from Step 8ci.

d. Processing is complete, and the External User Field is $s$.

9. Set $x$ to $x$-54,080,000,000.

10. If $x$ is less than 140,608,000,000, then perform the steps a-d below.

a. Define a character string, $s$, with an initial value of an empty string.

b. Define a character string, $t$, with a value of "FFNFNLLNL".

c. For each character, $c$, in $t$, starting from the right-most character, perform the steps i-iv below.

i. Divide $x$ by the number of allowed characters for character type abbreviation $c$ as specified in Table 3.

ii. Use the remainder from Step 10ci as an index into the allowed character values for the character type abbreviation $c$. (See Table 3.)

iii. Set $s$ to the concatenation of the indexed character from Step 10cii and $s$. (In other words, stick the indexed character onto the left side of the character string, $s$.)

iv. Set $x$ to the quotient from Step 10ci.

d. Processing is complete, and the External User Field is $s$.

11. Set $x$ to $x$-140,608,000,000.

12. If $x$ is less than 208,000,000, then perform the steps a-d below.

a. Define a character string, $s$, with an initial value of "  " (two space characters).

b. Define a character string, $t$, with a value of "FNNLLNLSS".

c. For each character, $c$, in $t$, starting from the right-most character, perform the steps i-iv below.

i. Divide $x$ by the number of allowed characters for character type abbreviation $c$ as specified in Table 3.

ii. Use the remainder from Step 12ci as an index into the allowed character values for the character type abbreviation $c$. (See Table 3.)

iii. Set $s$ to the concatenation of the indexed character from Step 12cii and $s$. (In other words, stick the indexed character onto the left side of the character string, $s$.)

iv. Set $x$ to the quotient from Step 12ci.

d. Processing is complete, and the External User Field is $s$.

13. Set $x$ to $x$-208,000,000.

14. Perform the steps a-d below.

a. Define a character string, $s$, with an initial value of " " (one space character).

b. Define a character string, $t$, with a value of “FNNNLLNL”.

c. For each character, $c$, in $t$, starting from the right-most character, perform the steps i-iv below.

    i. Divide $x$ by the number of allowed characters for character type abbreviation $c$ as specified in Table 3.

    ii. Use the remainder from Step 14ci as an index into the allowed character values for the character type abbreviation $c$. (See Table 3.)

    iii. Set $s$ to the concatenation of the indexed character from Step 14cii and $s$. (In other words, stick the indexed character onto the left side of the character string, $s$.)

    iv. Set $x$ to the quotient from Step 14ci.

d. Processing is complete, and the External User Field is $s$.

## 3.3.8 Conversion from External User Fields to Application String

The Application String is constructed by concatenating the five External User Fields in the order specified in Table 1, yielding a 26-character string.

# 3.4 Decoding Examples

The first two examples are counterparts of the two examples described in Section 2.3, “Encoding Examples”. The examples in this section concern the same data, but describe decoding, as opposed to encoding. The third example is a variation on Example 2 that involves incorrect bar identifications.

Each example shows the decoding of only one string of Bar Identifiers. Note that a specific decoding implementation will likely process multiple inputs for the same bar code, as described in Section 3.1, “Decoding Considerations.”

## 3.4.1 Example 1

Bar Identifiers:
“TTDTTATDDTTATTDTAATTDTAATDDTTATTDTTDATFTAATDDTAATDDTATATFAADDAATAATD
DTAADFTFTA”

| Extender Group Name | Bar Identifiers | Extender Group High Bits | Extender Group Low Bits | Extender Group Value | Data/Check Number | Data/Check Symbol Name |
|---|---|---|---|---|---|---|
| $E_0$ | TTD | $000_2$ | $001_2$ | $000001_2$ | 0 | $D_{11}$ |
| $E_1$ | TTA | $000_2$ | $001_2$ | $000001_2$ | 0 | $D_{12}$ |
| $E_2$ | TDD | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_0$ |
| $E_3$ | TTA | $000_2$ | $001_2$ | $000001_2$ | 0 | $D_{13}$ |
| $E_4$ | TTD | $000_2$ | $001_2$ | $000001_2$ | 0 | $D_{14}$ |
| $E_5$ | TAA | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_1$ |
| $E_6$ | TTD | $000_2$ | $001_2$ | $000001_2$ | 0 | $D_{15}$ |
| $E_7$ | TAA | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_2$ |
| $E_8$ | TDD | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_3$ |
| $E_9$ | TTA | $000_2$ | $001_2$ | $000001_2$ | 0 | $D_{16}$ |
| $E_{10}$ | TTD | $000_2$ | 001 | 000001 | 0 | $D_{17}$ |
| $E_{11}$ | TTD | $001_2$ | $000_2$ | $001000_2$ | 4 | $D_{18}$ |
| $E_{12}$ | ATF | $101_2$ | $001_2$ | $101001_2$ | 20 | $C_0$ |
| $E_{13}$ | TAA | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_4$ |
| $E_{14}$ | TDD | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_5$ |
| $E_{15}$ | TAA | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_6$ |
| $E_{16}$ | TDD | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_7$ |
| $E_{17}$ | TAT | $000_2$ | $010_2$ | $000010_2$ | 1 | $C_1$ |
| $E_{18}$ | ATF | $101_2$ | $001_2$ | $101001_2$ | 20 | $C_2$ |
| $E_{19}$ | AAD | $001_2$ | $110_2$ | $001110_2$ | 7 | $C_3$ |
| $E_{20}$ | DAA | 011 | $100_2$ | $011100_2$ | 14 | $C_4$ |
| $E_{21}$ | TAA | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_8$ |
| $E_{22}$ | TDD | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_9$ |
| $E_{23}$ | TAA | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_{10}$ |
| $E_{24}$ | DFT | $010_2$ | $110_2$ | $010110_2$ | 11 | $C_5$ |
| $E_{25}$ | FTA | $100_2$ | $101_2$ | $100101_2$ | 18 | $C_6$ |

**Table 16. Bar Identifiers to Extender Groups to Data/Check Symbols, Example 1.**

| Data Name | Data/Check Number Before Error Correction | Data/Check Number After Error Correction | Data Number After Error |
|---|---|---|---|
| $D_0$ | 0 | 0 | |
| $D_1$ | 0 | 0 | 0 |
| $D_2$ | 0 | 0 | 0 |
| $D_3$ | 0 | 0 | 0 |
| $D_4$ | 0 | 0 | 0 |
| $D_5$ | 0 | 0 | 0 |
| $D_6$ | 0 | 0 | 0 |
| $D_7$ | 0 | 0 | 0 |
| $D_8$ | 0 | 0 | 0 |
| $D_9$ | 0 | 0 | 0 |
| $D_{10}$ | 0 | 0 | 0 |
| $D_{11}$ | 0 | 0 | 0 |
| $D_{12}$ | 0 | 0 | 0 |
| $D_{13}$ | 0 | 0 | 0 |
| $D_{14}$ | 0 | 0 | 0 |
| $D_{15}$ | 0 | 0 | 0 |
| $D_{16}$ | 0 | 0 | 0 |
| $D_{17}$ | 0 | 0 | 0 |
| $D_{18}$ | 4 | 4 | 4 |
| $C_0$ | 20 | 20 | |
| $C_1$ | 1 | 1 | |
| $C_2$ | 20 | 20 | |
| $C_3$ | 7 | 7 | |
| $C_4$ | 14 | 14 | |
| $C_5$ | 11 | 11 | |
| $C_6$ | 18 | 18 | |

**Table 17. Data Numbers Before and After Reed-Solomon Error Correction, Example 1.**

Consolidated Data Value: 4

| Field Name | Internal User Field | External User Field |
|---|---|---|
| Format | 1 | “1” |
| Version ID | 0 | “1” |
| Class | 0 | “0” |
| Supply Chain ID | 0 | “000000” |
| Item ID | 0 | “00000000” |
| Destination Post Code plus DPS | 0 | “XY11 “ |

**Table 18. Internal User Fields to External User Fields, Example 1.**

Application String: “11000000000000000XY11 ”

Note that the Application String has five space characters at the end.

# 3.4.2 Example 2

Bar Identifiers:
“DTTFATTDDTATTTATFTDFFFTFDFDAFTTTADTTFDTFDDDTDFDDFTFAADTFDTDTDTFAATAF
DDTAATTDTT”

| Extender Group Name | Bar Identifiers | Extender Group High Bits | Extender Group Low Bits | Extender Group Value | Data/Check Number | Check Symbol |
|---|---|---|---|---|---|---|
| $E_0$ | DTT | $000_2$ | $100_2$ | $000100_2$ | 2 | $D_{11}$ |
| $E_1$ | FAT | $100_2$ | $110_2$ | $100110_2$ | 19 | $D_{12}$ |
| $E_2$ | TDD | $000_2$ | $011_2$ | $000011_2$ | 0 | $D_0$ |
| $E_3$ | TAT | $000_2$ | $010_2$ | $000010_2$ | 1 | $D_{13}$ |
| $E_4$ | TTA | $001_2$ | $000_2$ | $001000_2$ | 4 | $D_{14}$ |
| $E_5$ | TFT | $010_2$ | $010_2$ | $010010_2$ | 8 | $D_1$ |
| $E_6$ | DFF | $011_2$ | $111_2$ | $011111_2$ | 15 | $D_{15}$ |
| $E_7$ | FTF | $101_2$ | $101_2$ | $101101_2$ | 21 | $D_2$ |
| $E_8$ | DFD | $010_2$ | $111_2$ | $010111_2$ | 10 | $D_3$ |
| $E_9$ | AFT | $010_2$ | $110_2$ | $010110_2$ | 11 | $D_{16}$ |
| $E_{10}$ | TTA | $001_2$ | $000_2$ | $001000_2$ | 4 | $D_{17}$ |
| $E_{11}$ | DTT | $100_2$ | $000_2$ | $100000_2$ | 16 | $D_{18}$ |
| $E_{12}$ | FDT | $100_2$ | $110_2$ | $100110_2$ | 19 | $C_0$ |
| $E_{13}$ | FDD | $111_2$ | $100_2$ | $111100_2$ | 29 | $D_4$ |
| $E_{14}$ | DTD | $000_2$ | $101_2$ | $000101_2$ | 1 | $D_5$ |
| $E_{15}$ | FDD | $111_2$ | $100_2$ | $111100_2$ | 29 | $D_6$ |
| $E_{16}$ | FTF | $101_2$ | $101_2$ | $101101_2$ | 21 | $D_7$ |
| $E_{17}$ | AAD | $001_2$ | $110_2$ | $001110_2$ | 7 | $C_1$ |
| $E_{18}$ | TFD | $010_2$ | $011_2$ | $010011_2$ | 9 | $C_2$ |
| $E_{19}$ | TDT | $010_2$ | $000_2$ | $010000_2$ | 8 | $C_3$ |
| $E_{20}$ | DTF | $001_2$ | $101_2$ | $001101_2$ | 6 | $C_4$ |
| $E_{21}$ | AAT | $000_2$ | $110_2$ | $000110_2$ | 2 | $D_8$ |
| $E_{22}$ | AFD | $110_2$ | $011_2$ | $110011_2$ | 24 | $D_9$ |
| $E_{23}$ | DTA | $100_2$ | $001_2$ | $100001_2$ | 15 | $D_{10}$ |
| $E_{24}$ | ATT | $100_2$ | $000_2$ | $100000_2$ | 16 | $C_5$ |
| $E_{25}$ | DTT | $100_2$ | $000_2$ | $100000_2$ | 16 | $C_6$ |

**Table 19. Bar Identifiers to Extender Groups to Data/Check Symbols, Example 2.**

| Data Name | Data/Check Number Before Error Correction | Data/Check Number After Error Correction | Data Number After Error Correction |
|---|---|---|---|
| $D_0$ | 0 | 0 | 0 |
| $D_1$ | 8 | 8 | 8 |
| $D_2$ | 21 | 21 | 21 |
| $D_3$ | 10 | 10 | 10 |
| $D_4$ | 29 | 29 | 29 |
| $D_5$ | 1 | 1 | 1 |
| $D_6$ | 29 | 29 | 29 |
| $D_7$ | 21 | 21 | 21 |
| $D_8$ | 2 | 2 | 2 |
| $D_9$ | 24 | 24 | 24 |
| $D_{10}$ | 15 | 15 | 15 |
| $D_{11}$ | 2 | 2 | 2 |
| $D_{12}$ | 19 | 19 | 19 |
| $D_{13}$ | 1 | 1 | 1 |
| $D_{14}$ | 4 | 4 | 4 |
| $D_{15}$ | 15 | 15 | 15 |
| $D_{16}$ | 11 | 11 | 11 |
| $D_{17}$ | 4 | 4 | 4 |
| $D_{18}$ | 16 | 16 | 16 |
| $C_0$ | 19 | 19 | |
| $C_1$ | 7 | 7 | |
| $C_2$ | 9 | 9 | |
| $C_3$ | 8 | 8 | |
| $C_4$ | 6 | 6 | |
| $C_5$ | 16 | 16 | |
| $C_6$ | 16 | 16 | |

**Table 20. Data Numbers Before and After Reed-Solomon Error Correction, Example 2.**

Consolidated Data Value: 188,546,441,254,969,128,715,267,216

| Field Name | Internal User Field | External User Field |
|---|---|---|
| Format | 4 | “4” |
| Version ID | 0 | “1” |
| Class | 0 | “0” |
| Supply Chain ID | 384,224 | “384224” |
| Item ID | 16,563,762 | “16563762” |
| Destination Post Code plus DPS | 6,284,881,375 | “EF61AH8T “ |

**Table 21. Internal User Fields to External User Fields, Example 2.**

Application String: “41038422416563762EF61AH8T ”

Note that the Application String has one space character at the end.

# 3.4.3 Example 3

Bar Identifiers:
"**E**TTFAT**A**DDTAT**A**TATFT**E**FFFTF**E**FDAFT**A**TADTTFDTFDDDTDFDDFTFAADTFDTDTDTFAATAF DDTAATTDTT"Note that the three emphasised Bar Identifiers are incorrect.

| Extender Group Name | Bar Identifiers | Extender Group High Bits | Extender Group Low Bits | Extender Group Value | Data/Check Number | Data/Check Symbol Name |
|---|---|---|---|---|---|---|
| $\boldsymbol{E_0}$ | **ETT** | $\mathbf{000_2}$ | $\mathbf{100_2}$ | $\mathbf{000100_2}$ | **-1** | $\boldsymbol{D_{11}}$ |
| $E_1$ | FAT | $100_2$ | $110_2$ | $100110_2$ | 19 | $D_{12}$ |
| $\boldsymbol{E_2}$ | **ADD** | $\mathbf{000_2}$ | $\mathbf{011_2}$ | $\mathbf{000011_2}$ | **-1** | $\boldsymbol{D_0}$ |
| $E_3$ | TAT | $000_2$ | $010_2$ | $000010_2$ | 1 | $D_{13}$ |
| $\boldsymbol{E_4}$ | **ATA** | $\mathbf{001_2}$ | $\mathbf{000_2}$ | $\mathbf{001000_2}$ | **-1** | $\boldsymbol{D_{14}}$ |
| $E_5$ | TFT | $010_2$ | $010_2$ | $010010_2$ | 8 | $D_1$ |
| $\boldsymbol{E_6}$ | **EFF** | $\mathbf{011_2}$ | $\mathbf{111_2}$ | $\mathbf{011111_2}$ | **-1** | $\boldsymbol{D_{15}}$ |
| $E_7$ | FTF | $101_2$ | $101_2$ | $101101_2$ | 21 | $D_2$ |
| $\boldsymbol{E_8}$ | **EFD** | $\mathbf{010_2}$ | $\mathbf{111_2}$ | $\mathbf{010111_2}$ | **-1** | $\boldsymbol{D_3}$ |
| $E_9$ | AFT | $010_2$ | $110_2$ | $010110_2$ | 11 | $D_{16}$ |
| $\boldsymbol{E_{10}}$ | **ATA** | $\mathbf{001_2}$ | $\mathbf{000_2}$ | $\mathbf{001000_2}$ | **-1** | $\boldsymbol{D_{17}}$ |
| $E_{11}$ | DTT | $100_2$ | $000_2$ | $100000_2$ | 16 | $D_{18}$ |
| $E_{12}$ | FDT | $100_2$ | $110_2$ | $100110_2$ | 19 | $C_0$ |
| $E_{13}$ | FDD | $111_2$ | $100_2$ | $111100_2$ | 29 | $D_4$ |
| $E_{14}$ | DTD | $000_2$ | $101_2$ | $000101_2$ | 1 | $D_5$ |
| $E_{15}$ | FDD | $111_2$ | $100_2$ | $111100_2$ | 29 | $D_6$ |
| $E_{16}$ | FTF | $101_2$ | $101_2$ | $101101_2$ | 21 | $D_7$ |
| $E_{17}$ | AAD | $001_2$ | $110_2$ | $001110_2$ | 7 | $C_1$ |
| $E_{18}$ | TFD | $010_2$ | $011_2$ | $010011_2$ | 9 | $C_2$ |
| $E_{19}$ | TDT | $010_2$ | $000_2$ | $010000_2$ | 8 | $C_3$ |
| $E_{20}$ | DTF | $001_2$ | $101_2$ | $001101_2$ | 6 | $C_4$ |
| $E_{21}$ | AAT | $000_2$ | $110_2$ | $000110_2$ | 2 | $D_8$ |
| $E_{22}$ | AFD | $110_2$ | $011_2$ | $110011_2$ | 24 | $D_9$ |
| $E_{23}$ | DTA | $100_2$ | $001_2$ | $100001_2$ | 15 | $D_{10}$ |
| $E_{24}$ | ATT | $100_2$ | $000_2$ | $100000_2$ | 16 | $C_5$ |
| $E_{25}$ | DTT | $100_2$ | $000_2$ | $100000_2$ | 16 | $C_6$ |

**Table 22. Bar Identifiers to Extender Groups to Data/Check Symbols, Example 3.**

| Data Name | Data/Check Number Before Error Correction | Data/Check Number After Error Correction | Data Number After Error Correction |
|---|---|---|---|
| $\boldsymbol{D_0}$ | **-1** | **0** | **0** |
| $D_1$ | 8 | 8 | 8 |
| $D_2$ | 21 | 21 | 21 |
| $\boldsymbol{D_3}$ | **-1** | **10** | **10** |
| $D_4$ | 29 | 29 | 29 |
| $D_5$ | 1 | 1 | 1 |
| $D_6$ | 29 | 29 | 29 |
| $D_7$ | 21 | 21 | 21 |
| $D_8$ | 2 | 2 | 2 |
| $D_9$ | 24 | 24 | 24 |
| $D_{10}$ | 15 | 15 | 15 |
| $\boldsymbol{D_{11}}$ | **-1** | **2** | **2** |
| $D_{12}$ | 19 | 19 | 19 |
| $D_{13}$ | 1 | 1 | 1 |
| $\boldsymbol{D_{14}}$ | **-1** | **4** | **4** |
| $\boldsymbol{D_{15}}$ | **-1** | **15** | **15** |
| $D_{16}$ | 11 | 11 | 11 |
| $\boldsymbol{D_{17}}$ | **-1** | **4** | **4** |
| $D_{18}$ | 16 | 16 | 16 |
| $C_0$ | 19 | 19 | |
| $C_1$ | 7 | 7 | |
| $C_2$ | 9 | 9 | |
| $C_3$ | 8 | 8 | |
| $C_4$ | 6 | 6 | |
| $C_5$ | 16 | 16 | |
| $C_6$ | 16 | 16 | |

**Table 23. Data Numbers Before and After Reed-Solomon Error Correction, Example 3.**

Consolidated Data Value: 188,546,441,254,969,128,715,267,216

| Field Name | Internal User Field | External User Field |
|---|---|---|
| Format | 4 | “4” |
| Version ID | 0 | “1” |
| Class | 0 | “0” |
| Supply Chain ID | 384,224 | “384224” |
| Item ID | 16,563,762 | “16563762” |
| Destination Post Code plus DPS | 6,284,881,375 | “EF61AH8T “ |

**Table 24. Internal User Fields to External User Fields, Example 3.**

Application String: “41038422416563762EF61AH8T ”

Note that the Application String has one space character at the end.

# 4 Referenced Documents

The following documents are referenced within this document or provide additional reading that augments the content of this document.

| Reference | Document Name | Document Number |
|---|---|---|
| 1 | Royal Mail Mailmark barcode definition document 1st September 2015.doc | Based on internal version 4.0 of 2012-EIB-000251 |

**Table 25. Referenced Documents**

# 5 List of Acronyms

| Acronym | Description |
|---|---|
| DPS | Delivery Point Suffix |
| EIB® | Enterprise Intelligent Barcode |
| ID | Identifier |

**Table 26. List of Acronyms**

# END