Stream:	Internet Engineering Task Force (IETF)
RFC:	9285
Category:	Informational
Published:	July 2022
ISSN:	2070-1721

Authors:	P. Fältström	F. Ljunggren	D. van Gulik
	Netnod	*Kirei*	*Webweaving*

RFC 9285
The Base45 Data Encoding

Abstract

This document describes the Base45 encoding scheme, which is built upon the Base64, Base32, and Base16 encoding schemes.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9285.

Copyright Notice

Table of Contents

1. Introduction

A QR code is used to encode text as a graphical image. Depending on the characters used in the text, various encoding options for a QR code exist, e.g., Numeric, Alphanumeric, and Byte mode. Even in Byte mode, a typical QR code reader tries to interpret a byte sequence as text encoded in UTF-8 or ISO/IEC 8859-1. Thus, QR codes cannot be used to encode arbitrary binary data directly. Such data has to be converted into an appropriate text before that text could be encoded as a QR code. Compared to already established Base64, Base32, and Base16 encoding schemes that are described in [RFC4648], the Base45 scheme described in this document offers a more compact QR code encoding.

One important difference from those others and Base45 is the key table and that the padding with '=' is not required.

2. Conventions Used in This Document

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Interpretation of Encoded Data

Encoded data is to be interpreted as described in [RFC4648] with the exception that a different alphabet is selected.

4. The Base45 Encoding

QR codes have a limited ability to store binary data. In practice, binary data have to be encoded in characters according to one of the modes already defined in the standard for QR codes. The easiest mode to use in called Alphanumeric mode (see Section 7.3.4 and Table 2 of [ISO18004]. Unfortunately Alphanumeric mode uses 45 different characters which implies neither Base32 nor Base64 are very effective encodings.

A 45-character subset of US-ASCII is used; the 45 characters usable in a QR code in Alphanumeric mode (see Section 7.3.4 and Table 2 of [ISO18004]). Base45 encodes 2 bytes in 3 characters, compared to Base64 that encodes 3 bytes in 4 characters.

For encoding, two bytes [a, b] **MUST** be interpreted as a number n in Base256, i.e. as an unsigned integer over 16 bits so that the number n = (a*256) + b.

This number n is converted to Base45 [c, d, e] so that n = c + (d*45) + (e*45*45). Note the order of c, d and e which are chosen so that the left-most [c] is the least significant.

The values c, d and e are then looked up in Table 1 to produce a three character string. The process is reversed when decoding.

For encoding a single byte [a], it **MUST** be interpreted as a Base256 number, i.e. as an unsigned integer over 8 bits. That integer **MUST** be converted to Base45 [c d] so that a = c + (45*d). The values c and d are then looked up in Table 1 to produce a two-character string.

A byte string [a b c d ... x y z] with arbitrary content and arbitrary length **MUST** be encoded as follows: From left to right pairs of bytes **MUST** be encoded as described above. If the number of bytes is even, then the encoded form is a string with a length that is evenly divisible by 3. If the number of bytes is odd, then the last (rightmost) byte **MUST** be encoded on two characters as described above.

For decoding a Base45 encoded string the inverse operations are performed.

4.1. When to, and not to, use Base45

If binary data is to be stored in a QR code, the suggested mechanism is to use the Alphanumeric mode that uses 11 bits for 2 characters as defined in Section 7.3.4 of [ISO18004]. The ECI mode indicator for this encoding is 0010.

On the other hand if the data is to be sent via some other transport, a transport encoding suitable for that transport should be used instead of Base45. For example, it is not recommended to first encode data in Base45 and then encode the resulting string in Base64 if the data is to be sent via email. Instead, the Base45 encoding should be removed, and the data itself should be encoded in Base64.

4.2. The Alphabet Used in Base45

The Alphanumeric mode is defined to use 45 characters as specified in this alphabet.

Value Encoding	Value Encoding	Value Encoding	Value Encoding
00 0	12 C	24 O	36 Space
01 1	13 D	25 P	37 $
02 2	14 E	26 Q	38 %
03 3	15 F	27 R	39 *
04 4	16 G	28 S	40 +
05 5	17 H	29 T	41 -
06 6	18 I	30 U	42 .
07 7	19 J	31 V	43 /
08 8	20 K	32 W	44 :
09 9	21 L	33 X	
10 A	22 M	34 Y	
11 B	23 N	35 Z	

Table 1: The Base45 Alphabet

4.3. Encoding Examples

It should be noted that although the examples are all text, Base45 is an encoding for binary data where each octet can have any value 0-255.

Option (a): added line breaks

Encoding example 1:

The string is "AB".

As ASCII, the byte sequence is [65 66].

The 16-bit value is 65 * 256 + 66 = 16706.

16706 equals 11 + (11 * 45) + (8 * 45 * 45), so in Base45, it is [11 11 8].

Referring to Table 1, the encoded string is "BB8".

Encoding example 2:

The string is "Hello!!".

As ASCII, the byte sequence is [72 101 108 108 111 33 33].

The 16-bit values are [18533 27756 28449 33].

In Base45, it is [[38 6 9] [36 31 13] [9 2 14] [33 0]]. Note the 33 for the last byte.

Referring to Table 1, the encoded string is "%69 VD92EX0". Note that it includes a space.

Encoding example 3:

The string is "base-45".

As ASCII, the byte sequence is [98 97 115 101 45 52 53].

The 16-bit values are [25185 29541 11572 53].

In Base45, it is [[30 19 12] [21 26 14] [7 32 5] [8 1]].

Referring to Table 1, the encoded string is "UJCLQE7W581".

4.4. Decoding Examples

Decoding example 1:

The encoded string is "QED8WEX0".

Referring to Table 1, this represents [26 14 13 8 32 14 33 0].

Arranging the numbers in chunks of three, except for the last one which can be two numbers, we get [[26 14 13] [8 32 14] [33 0]].

In Base45, we get [26981 29798 33].

The byte sequence is [[105 101] [116 102] [33]].

The ASCII is string "ietf!".

5. IANA Considerations

This document has no IANA actions.

6. Security Considerations

When implementing encoding and decoding it is important to be very careful so that buffer overflow or similar does not occur. This of course includes the calculations in Base45 and lookup in the table of characters (Table 1). A decoder must also be robust regarding input, including proper handling of any octet value 0-255, including the NUL character (ASCII 0).

It should be noted that Base64 and some other encodings pad the string so that the encoding starts with an aligned number of characters while Base45 specifically avoids padding. Because of this, special care has to be taken when odd numbers of octets are to be encoded. Similarly, care must be taken if the number of characters to decode are not evenly divisible by 3.

Base encodings use a specific, reduced alphabet to encode binary data. Non-alphabet characters could exist within base-encoded data, caused by data corruption or by design. Non-alphabet characters may be exploited as a "covert channel", where non-protocol data can be sent for nefarious purposes. Non-alphabet characters might also be sent in order to exploit implementation errors leading to, for example, buffer overflow attacks.

Implementations **MUST** reject any input that is not a valid encoding. For example, it **MUST** reject the input (encoded data) if it contains characters outside the base alphabet (in Table 1) when interpreting base-encoded data.

Even though a Base45-encoded string contains only characters from the alphabet in Table 1, cases like the following have to be considered: The string "FGW" represents 65535 (FFFF in Base16), which is a valid encoding of 16 bits. A slightly different encoded string of the same length, "GGW", would represent 65536 (10000 in Base16), which is represented by more than 16 bits. Implementations **MUST** also reject the encoded data if it contains a triplet of characters that, when decoded, results in an unsigned integer that is greater than 65535 (ffff in Base16).

It should be noted that the resulting string after encoding to Base45 might include non-URL-safe characters so if the URL including the Base45 encoded data has to be URL-safe, one has to use percent-encoding.

7. Normative References

[ISO18004] ISO/IEC, "Information technology - Automatic identification and data capture techniques - QR Code bar code symbology specification", ISO/IEC 18004:2015, February 2015, <https://www.iso.org/standard/62021.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Acknowledgements

The authors thank Mark Adler, Anders Ahl, Alan Barrett, Sam Spens Clason, Alfred Fiedler, Tomas Harreveld, Erik Hellman, Joakim Jardenberg, Michael Joost, Erik Kline, Christian Landgren, Anders Lowinger, Mans Nilsson, Jakob Schlyter, Peter Teufl, and Gaby Whitehead for the feedback. Also, everyone who has been working with Base64 over a long period of years and has proven the implementations are stable.

Authors' Addresses

Patrik Fältström
Netnod
Email: paf@netnod.se

Fredrik Ljunggren
Kirei
Email: fredrik@kirei.se

Dirk-Willem van Gulik
Webweaving
Email: dirkx@webweaving.org