Network Working Group
Request for Comments: 3548
Category: Informational

S. Josefsson, Ed.

July 2003

The Base16, Base32, and Base64 Data Encodings

Status of this Memo

Copyright Notice

Abstract

This document describes the commonly used base 64, base 32, and base 16 encoding schemes. It also discusses the use of line-feeds in encoded data, use of padding in encoded data, use of non-alphabet characters in encoded data, and use of different encoding alphabets.

Table of Contents

1. Introduction

Base encoding of data is used in many situations to store or transfer data in environments that, perhaps for legacy reasons, are restricted to only US-ASCII [9] data. Base encoding can also be used in new applications that do not have legacy restrictions, simply because it makes it possible to manipulate objects with text editors.

In the past, different applications have had different requirements and thus sometimes implemented base encodings in slightly different ways. Today, protocol specifications sometimes use base encodings in general, and "base64" in particular, without a precise description or reference. MIME [3] is often used as a reference for base64 without considering the consequences for line-wrapping or non-alphabet characters. The purpose of this specification is to establish common alphabet and encoding considerations. This will hopefully reduce ambiguity in other documents, leading to better interoperability.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

2. Implementation discrepancies

Here we discuss the discrepancies between base encoding implementations in the past, and where appropriate, mandate a specific recommended behavior for the future.

2.1. Line feeds in encoded data

MIME [3] is often used as a reference for base 64 encoding. However, MIME does not define "base 64" per se, but rather a "base 64 Content-Transfer-Encoding" for use within MIME. As such, MIME enforces a limit on line length of base 64 encoded data to 76 characters. MIME inherits the encoding from PEM [2] stating it is "virtually identical", however PEM uses a line length of 64 characters. The MIME and PEM limits are both due to limits within SMTP.

Implementations MUST NOT not add line feeds to base encoded data unless the specification referring to this document explicitly directs base encoders to add line feeds after a specific number of characters.

2.2. Padding of encoded data

In some circumstances, the use of padding ("=") in base encoded data is not required nor used. In the general case, when assumptions on size of transported data cannot be made, padding is required to yield correct decoded data.

Implementations MUST include appropriate pad characters at the end of encoded data unless the specification referring to this document explicitly states otherwise.

2.3. Interpretation of non-alphabet characters in encoded data

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, e.g., buffer overflow attacks.

Implementations MUST reject the encoding if it contains characters outside the base alphabet when interpreting base encoded data, unless the specification referring to this document explicitly states otherwise. Such specifications may, as MIME does, instead state that characters outside the base encoding alphabet should simply be ignored when interpreting data ("be liberal in what you accept"). Note that this means that any CRLF constitute "non alphabet characters" and are ignored. Furthermore, such specifications may consider the pad character, "=", as not part of the base alphabet until the end of the string. If more than the allowed number of pad characters are found at the end of the string, e.g., a base 64 string terminated with "===", the excess pad characters could be ignored.

2.4. Choosing the alphabet

Different applications have different requirements on the characters in the alphabet. Here are a few requirements that determine which alphabet should be used:

- Handled by humans. Characters "0", "O" are easily interchanged, as well "1", "l" and "I". In the base32 alphabet below, where 0 (zero) and 1 (one) is not present, a decoder may interpret 0 as O, and 1 as I or L depending on case. (However, by default it should not, see previous section.)

- Encoded into structures that place other requirements. For base 16 and base 32, this determines the use of upper- or lowercase alphabets. For base 64, the non-alphanumeric characters (in particular "/") may be problematic in file names and URLs.

- Used as identifiers. Certain characters, notably "+" and "/" in the base 64 alphabet, are treated as word-breaks by legacy text search/index tools.

There is no universally accepted alphabet that fulfills all the requirements. In this document, we document and name some currently used alphabets.

3. Base 64 Encoding

The following description of base 64 is due to [2], [3], [4] and [5].

The Base 64 encoding is designed to represent arbitrary sequences of octets in a form that requires case sensitivity but need not be humanly readable.

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.)

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base 64 alphabet.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string.

Table 1: The Base 64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Special processing is performed if fewer than 24 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a quantity. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the '=' character. Since all base 64 input is an integral number of octets, only the following cases can arise:

(1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding,

(2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or

(3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

4. Base 64 Encoding with URL and Filename Safe Alphabet

The Base 64 encoding with an URL and filename safe alphabet has been used in [8].

An alternative alphabet has been suggested that used "~" as the 63rd character. Since the "~" character has special meaning in some file system environments, the encoding described in this section is recommended instead.

This encoding should not be regarded as the same as the "base64" encoding, and should not be referred to as only "base64". Unless made clear, "base64" refer to the base 64 in the previous section.

This encoding is technically identical to the previous one, except for the 62:nd and 63:rd alphabet character, as indicated in table 2.

Table 2: The "URL and Filename safe" Base 64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	- (minus)
12	M	29	d	46	u	63	_ (understrike)
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

5. Base 32 Encoding

The following description of base 32 is due to [7] (with corrections).

The Base 32 encoding is designed to represent arbitrary sequences of octets in a form that needs to be case insensitive but need not be humanly readable.

A 33-character subset of US-ASCII is used, enabling 5 bits to be represented per printable character. (The extra 33rd character, "=", is used to signify a special processing function.)

The encoding process represents 40-bit groups of input bits as output strings of 8 encoded characters. Proceeding from left to right, a 40-bit input group is formed by concatenating 5 8bit input groups. These 40 bits are then treated as 8 concatenated 5-bit groups, each of which is translated into a single digit in the base 32 alphabet. When encoding a bit stream via the base 32 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first 8bit byte, and the eighth bit will be the low-order bit in the first 8bit byte, and so on.

Each 5-bit group is used as an index into an array of 32 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 2, below, are selected from US-ASCII digits and uppercase letters.

Table 3: The Base 32 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y	(pad)	=
7	H	16	Q	25	Z		
8	I	17	R	26	2		

Special processing is performed if fewer than 40 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 40 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 5-bit groups. Padding at the end of the data is performed using the "=" character. Since all base 32 input is an integral number of octets, only the following cases can arise:

(1) the final quantum of encoding input is an integral multiple of 40 bits; here, the final unit of encoded output will be an integral multiple of 8 characters with no "=" padding,

(2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by six "=" padding characters,

(3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be four characters followed by four "=" padding characters,

(4) the final quantum of encoding input is exactly 24 bits; here, the final unit of encoded output will be five characters followed by three "=" padding characters, or

(5) the final quantum of encoding input is exactly 32 bits; here, the final unit of encoded output will be seven characters followed by one "=" padding character.

6. Base 16 Encoding

The following description is original but analogous to previous descriptions. Essentially, Base 16 encoding is the standard standard case insensitive hex encoding, and may be referred to as "base16" or "hex".

A 16-character subset of US-ASCII is used, enabling 4 bits to be represented per printable character.

The encoding process represents 8-bit groups (octets) of input bits as output strings of 2 encoded characters. Proceeding from left to right, a 8-bit input is taken from the input data. These 8 bits are then treated as 2 concatenated 4-bit groups, each of which is translated into a single digit in the base 16 alphabet.

Each 4-bit group is used as an index into an array of 16 printable characters. The character referenced by the index is placed in the output string.

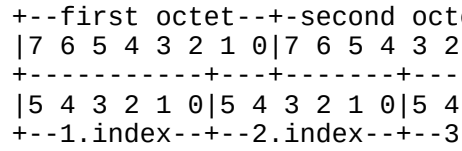
Table 5: The Base 16 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	0	4	4	8	8	12	C
1	1	5	5	9	9	13	D
2	2	6	6	10	A	14	E
3	3	7	7	11	B	15	F

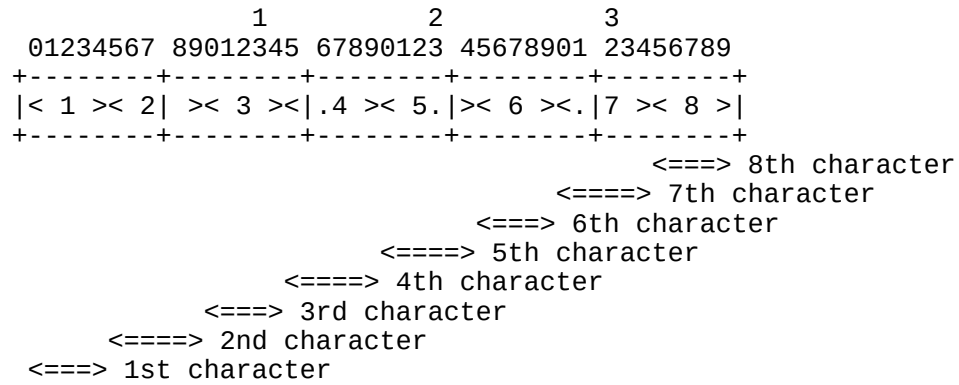
Unlike base 32 and base 64, no special padding is necessary since a full code word is always available.

7. Illustrations and examples

To translate between binary and a base encoding, the input is stored in a structure and the output is extracted. The case for base 64 is displayed in the following figure, borrowed from [4].



The case for base 32 is shown in the following figure, borrowed from [6]. Each successive character in a base-32 value represents 5 successive bits of the underlying octet sequence. Thus, each group of 8 characters represents a sequence of 5 octets (40 bits).



The following example of Base64 data is from [4].

```
    Input data:  0x14fb9c03d97e
    Hex:     1   4    f   b    9   c     | 0   3    d   9    7   e
    8-bit:   00010100 11111011 10011100  | 00000011 11011001
    11111110
    6-bit:   000101 001111 101110 011100 | 000000 111101 100111
    111110
    Decimal: 5      15     46     28       0      61     37     62
    Output:  F      P      u      c        A      9      l      +

    Input data:  0x14fb9c03d9
    Hex:     1   4    f   b    9   c     | 0   3    d   9
    8-bit:   00010100 11111011 10011100  | 00000011 11011001
                                                    pad with 00
    6-bit:   000101 001111 101110 011100 | 000000 111101 100100
    Decimal: 5      15     46     28       0      61     36
                                                         pad with =
    Output:  F      P      u      c        A      9      k      =

    Input data:  0x14fb9c03
    Hex:     1   4    f   b    9   c     | 0   3
    8-bit:   00010100 11111011 10011100  | 00000011
                                           pad with 0000
    6-bit:   000101 001111 101110 011100 | 000000 110000
    Decimal: 5      15     46     28       0      48
                                                pad with =      =
    Output:  F      P      u      c        A      w      =      =
```

8. Security Considerations

When implementing Base encoding and decoding, care should be taken not to introduce vulnerabilities to buffer overflow attacks, or other attacks on the implementation. A decoder should not break on invalid input including, e.g., embedded NUL characters (ASCII 0).

If non-alphabet characters are ignored, instead of causing rejection of the entire encoding (as recommended), a covert channel that can be used to "leak" information is made possible. The implications of this should be understood in applications that do not follow the recommended practice. Similarly, when the base 16 and base 32 alphabets are handled case insensitively, alteration of case can be used to leak information.

Base encoding visually hides otherwise easily recognized information, such as passwords, but does not provide any computational confidentiality. This has been known to cause security incidents when, e.g., a user reports details of a network protocol exchange

(perhaps to illustrate some other problem) and accidentally reveals the password because she is unaware that the base encoding does not protect the password.

9. References

9.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

9.2. Informative References

[2] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

[3] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[4] Callas, J., Donnerhacke, L., Finney, H. and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[5] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[6] Klyne, G. and L. Masinter, "Identifying Composite Media Features", RFC 2938, September 2000.

[7] Myers, J., "SASL GSSAPI mechanisms", Work in Progress.

[8] Wilcox-O'Hearn, B., "Post to P2P-hackers mailing list", World Wide Web http://zgp.org/pipermail/p2p-hackers/2001-September/000315.html, September 2001.

[9] Cerf, V., "ASCII format for Network Interchange", RFC 20, October 1969.

10. Acknowledgements

Several people offered comments and suggestions, including Tony Hansen, Gordon Mohr, John Myers, Chris Newman, and Andrew Sieber. Text used in this document is based on earlier RFCs describing specific uses of various base encodings. The author acknowledges the RSA Laboratories for supporting the work that led to this document.

11. Editor's Address

Simon Josefsson
EMail: simon@josefsson.org

12. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.