WIKIPEDIA

Base64

In programming, **Base64** is a group of binary-to-text encoding schemes that represent binary data (more specifically, a sequence of 8-bit bytes) in an ASCII string format by translating the data into a radix-64 representation. The term *Base64* originates from a specific MIME content transfer encoding. Each non-final Base64 digit represents exactly 6 bits of data. Three 8-bit bytes (i.e., a total of 24 bits) can therefore be represented by four 6-bit Base64 digits.

Common to all binary-to-text encoding schemes, Base64 is designed to carry data stored in binary formats across channels that only reliably support text content. Base64 is particularly prevalent on the World Wide Web[1] where its uses include the ability to embed image files or other binary assets inside textual assets such as HTML and CSS files.[2]

Base64 is also widely used for sending e-mail attachments. This is required because SMTP - in its original form - was designed to transport 7-bit ASCII characters only. This encoding causes an overhead of 33–36% (33% by the encoding itself; up to 3% more by the inserted line breaks).

Contents

Design

The particular set of 64 characters chosen to represent the 64-digit values for the base varies between implementations. The general strategy is to choose 64 characters that are common to most encodings and that are also printable. This combination leaves the data unlikely to be modified in transit through information systems, such as email, that were traditionally not 8-bit clean.[3] For example, MIME's Base64 implementation uses `A`–`Z`, `a`–`z`, and `0`–`9` for the first 62 values. Other variations share this property but differ in the symbols chosen for the last two values; an example is UTF-7.

The earliest instances of this type of encoding were created for dial-up communication between systems running the same OS — e.g., uuencode for UNIX, BinHex for the TRS-80 (later adapted for the Macintosh) — and could therefore make more assumptions about what characters were safe to use. For instance, uuencode uses uppercase letters, digits, and many punctuation characters, but no lowercase.[4][5][6][3]

Base64 table

The Base64 index table:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	`A`	16	010000	`Q`	32	100000	`g`	48	110000	`w`
1	000001	`B`	17	010001	`R`	33	100001	`h`	49	110001	`x`
2	000010	`C`	18	010010	`S`	34	100010	`i`	50	110010	`y`
3	000011	`D`	19	010011	`T`	35	100011	`j`	51	110011	`z`
4	000100	`E`	20	010100	`U`	36	100100	`k`	52	110100	`0`
5	000101	`F`	21	010101	`V`	37	100101	`l`	53	110101	`1`
6	000110	`G`	22	010110	`W`	38	100110	`m`	54	110110	`2`
7	000111	`H`	23	010111	`X`	39	100111	`n`	55	110111	`3`
8	001000	`I`	24	011000	`Y`	40	101000	`o`	56	111000	`4`
9	001001	`J`	25	011001	`Z`	41	101001	`p`	57	111001	`5`
10	001010	`K`	26	011010	`a`	42	101010	`q`	58	111010	`6`
11	001011	`L`	27	011011	`b`	43	101011	`r`	59	111011	`7`
12	001100	`M`	28	011100	`c`	44	101100	`s`	60	111100	`8`
13	001101	`N`	29	011101	`d`	45	101101	`t`	61	111101	`9`
14	001110	`O`	30	011110	`e`	46	101110	`u`	62	111110	`+`
15	001111	`P`	31	011111	`f`	47	101111	`v`	63	111111	`/`
Padding		`=`									

Examples

The example below uses ASCII text for simplicity, but this is not a typical use case, as it can already be safely transferred across all systems that can handle Base64. The more typical use is to encode binary data (such as an image); the resulting Base64 data will only contain 64 different ASCII characters, all of which can reliably be transferred across systems that may corrupt the raw source bytes.

Here is a quote from Thomas Hobbes's *Leviathan*:

```
Man is distinguished, not only by his reason, but by this singular passion from other animals,
which is a lust of the mind, that by a perseverance of delight in the continued and indefatigable
generation of knowledge, exceeds the short vehemence of any carnal pleasure.
```

(Please note that all of the example encodings below use only the bytes shown here; it is not a null-terminated string.)

When that quote is encoded into Base64, it is represented as a byte sequence of 8-bit-padded ASCII characters encoded in MIME's Base64 scheme as follows (newlines and white spaces may be present anywhere but are to be ignored on decoding):

```
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=
```

In the above quote, the encoded value of *Man* is *TWFu*. Encoded in ASCII, the characters *M*, *a*, and *n* are stored as the byte values `77`, `97`, and `110`, which are the 8-bit binary values `01001101`, `01100001`, and `01101110`. These three values are joined together into a 24-bit string, producing `010011010110000101101110`. Groups of 6 bits (6 bits have a maximum of $2^6 = 64$ different binary values) are converted into individual numbers from left to right (in this case, there are four numbers in a 24-bit string), which are then converted into their corresponding Base64 character values.

As this example illustrates, Base64 encoding converts three octets into four encoded characters.

Source	Text (ASCII)	M								a								n							
	Octets	77 (0x4d)								97 (0x61)								110 (0x6e)							
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Base64 encoded	Sextets	19						22						5						46					
	Character	T						W						F						u					
	Octets	84 (0x54)						87 (0x57)						70 (0x46)						117 (0x75)					

= padding characters might be added to make the last encoded block contain four Base64 characters.

Hexadecimal to octal transformation is useful to convert between binary and Base64. Both for advanced calculators and programming languages, such conversion is available. For example, the 24 bits above - when converted to hexadecimal - is 4D616E. Those 24 bits - when converted to octal - is 23260556. Those 8 octal digits - when divided into four groups - is 23 26 05 56. Each 2-digit group - when converted to decimal - is 19 22 05 46. Using those four decimal numbers as indices for the Base64 index table, the corresponding ASCII characters are *TWFu*.

If there are only two significant input octets (e.g., 'Ma'), or when the last input group contains only two octets, all 16 bits will be captured in the first three Base64 digits (18 bits); the two least significant bits of the last content-bearing 6-bit block will turn out to be zero, and discarded on decoding (along with the succeeding = padding character):

Source	Text (ASCII)	M								a															
	Octets	77 (0x4d)								97 (0x61)															
Bits		0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	0						
Base64 encoded	Sextets	19						22						4						Padding					
	Character	T						W						E						=					
	Octets	84 (0x54)						87 (0x57)						69 (0x45)						61 (0x3D)					

If there is only one significant input octet (e.g., 'M'), or when the last input group contains only one octet, all 8 bits will be captured in the first two Base64 digits (12 bits); the four least significant bits of the last content-bearing 6-bit block will turn out to be zero, and discarded on decoding (along with the succeeding two = padding characters):

Source	Text (ASCII)	M																							
	Octets	77 (0x4d)																							
Bits		0	1	0	0	1	1	0	1	0	0	0	0												
Base64 encoded	Sextets	19						16						Padding						Padding					
	Character	T						Q						=						=					
	Octets	84 (0x54)						81 (0x51)						61 (0x3D)						61 (0x3D)					

Output padding

Because Base64 is a six-bit encoding, and because the decoded values are divided into 8-bit octets on a modern computer, every four characters of Base64-encoded text (4 sextets = 4*6 = 24 bits) represents three octets of unencoded text or data (3 octets = 3*8 = 24 bits). This means that when the length of the unencoded input is not a multiple of three, the encoded output must have padding added so that its length is a multiple of four. The padding character is =, which indicates that no further bits are needed to fully encode the input. (This is different from `A`, which means that the remaining bits are all zeros.) The example below illustrates how truncating the input of the above quote changes the output padding:

Input		Output		Padding
Length	Text	Length	Text	
20	*any carnal pleasure.*	28	`YW55IGNhcm5hbCBwbGVhc3VyZS4=`	1
19	*any carnal pleasure*	28	`YW55IGNhcm5hbCBwbGVhc3VyZQ==`	2
18	*any carnal pleasur*	24	`YW55IGNhcm5hbCBwbGVhc3Vy`	0
17	*any carnal pleasu*	24	`YW55IGNhcm5hbCBwbGVhc3U=`	1
16	*any carnal pleas*	24	`YW55IGNhcm5hbCBwbGVhcw==`	2

The padding character is not essential for decoding, since the number of missing bytes can be inferred from the length of the encoded text. In some implementations, the padding character is mandatory, while for others it is not used. An exception in which padding characters are required is when multiple Base64 encoded files have been concatenated.

Another consequence of the sextet encoding of octets is that the same octet will be encoded differently depending on its position within a three-octet group of the input, and depending on which particular octet precedes it within the group. For example:

Input	Output
pleasure.	`cGxlYXN1cmUu`
leasure.	`bGVhc3VyZS4=`
easure.	`ZWFzdXJlLg==`
asure.	`YXN1cmUu`
sure.	`c3VyZS4=`

As the eight bits of an octet are spread across multiple sextets within the output, this is an obvious consequence, since no octet can be stuffed into a single sextet; instead, they must share.

However, since the sextets or characters of the output must be saved and manipulated on the same computer system, which only understands octets, they must be represented as octets, with the upper two bits set to zero. (In other words, the encoded output `YW55` occupies 4*8 = 32 bits, even though only 24 bits are meaningfully derived from the input, '*any*'.) Indeed, these supposedly wasted bits are exactly the reason for the Base64 encoding. The ratio of output bytes to input bytes is 4:3 (33% overhead). Specifically, given an input of n bytes, the output will be $4\left\lceil \frac{n}{3} \right\rceil$ bytes long, including padding characters.

Decoding Base64 with padding

When decoding Base64 text, four characters are typically converted back to three bytes. The only exceptions are when padding characters exist. A single = indicates that the four characters will decode to only two bytes, while == indicates that the four characters will decode to only a single byte. For example:

Encoded	Padding	Length	Decoded
`YW55IGNhcm5hbCBwbGVhcw==`	==	1	*any carnal pleas*
`YW55IGNhcm5hbCBwbGVhc3U=`	=	2	*any carnal pleasu*
`YW55IGNhcm5hbCBwbGVhc3Vy`	None	3	*any carnal pleasur*
`YW55IGNhcm5hbCBwbGVhc3VyZQ==`	==	1	*any carnal pleasure*

Decoding Base64 without padding

Without padding, after normal decoding of four characters to three bytes over and over again, fewer than four encoded characters may remain. In this situation, only two or three characters can remain. A single remaining encoded character is not possible (because a single Base64 character only contains 6 bits, and 8 bits are required to create a byte, so a minimum of two Base64 characters are required: The first character contributes 6 bits, and the second character contributes its first 2 bits.) For example:

Length	Encoded	Length	Decoded
2	`YW55IGNhcm5hbCBwbGVhcw`	1	*any carnal pleas*
3	`YW55IGNhcm5hbCBwbGVhc3U`	2	*any carnal pleasu*
4	`YW55IGNhcm5hbCBwbGVhc3Vy`	3	*any carnal pleasur*

Implementations and history

Variants summary table

Implementations may have some constraints on the alphabet used for representing some bit patterns. This notably concerns the last two characters used in the index table for index 62 and 63, and the character used for padding (which may be mandatory in some protocols or removed in others). The table below summarizes these known variants and provides links to the subsections below.

Encoding	Encoding characters			Separate encoding of lines			Decoding non-encoding characters
	62nd	63rd	*pad*	Separators	Length	Checksum	
RFC 1421: Base64 for Privacy-Enhanced Mail (deprecated)	+	/	= mandatory	CR+LF	64, or lower for the last line	No	No
RFC 2045: Base64 transfer encoding for MIME	+	/	= mandatory	CR+LF	At most 76	No	Discarded
RFC 2152: Base64 for UTF-7	+	/	No	No			No
RFC 3501 (https://tools.ietf.org/html/rfc3501#section-5.1.3): Base64 encoding for IMAP mailbox names	+	,	No	No			No
RFC 4648 §4 (https://tools.ietf.org/html/rfc4648#section-4): base64 (standard)[a]	+	/	= optional	No			No
RFC 4648 §5 (https://tools.ietf.org/html/rfc4648#section-5): base64url (URL- and filename-safe standard)[a]	-	_	= optional	No			No
RFC 4880: Radix-64 for OpenPGP	+	/	= mandatory	CR+LF	At most 76	Radix-64 encoded 24-bit CRC	No

a. It is important to note that this variant is intended to provide common features where they are not desired to be specialised by implementations, ensuring robust engineering. This is particularly in light of separate line encodings and restrictions, which have not been considered when previous standards have been co-opted for use elsewhere. Thus, the features indicated here may be over-ridden.

Privacy-enhanced mail

The first known standardized use of the encoding now called MIME Base64 was in the Privacy-enhanced Electronic Mail (PEM) protocol, proposed by RFC 989 (https://tools.ietf.org/html/rfc989) in 1987. PEM defines a "printable encoding" scheme that uses Base64 encoding to transform an arbitrary sequence of octets to a format that can be expressed in short lines of 6-bit characters, as required by transfer protocols such as SMTP.[7]

The current version of PEM (specified in RFC 1421 (https://tools.ietf.org/html/rfc1421)) uses a 64-character alphabet consisting of upper- and lower-case Roman letters (`A`–`Z`, `a`–`z`), the numerals (`0`–`9`), and the `+` and `/` symbols. The `=` symbol is also used as a padding suffix.[4] The original specification, RFC 989 (https://tools.ietf.org/html/rfc989), additionally used the `*` symbol to delimit encoded but unencrypted data within the output stream.

To convert data to PEM printable encoding, the first byte is placed in the most significant eight bits of a 24-bit buffer, the next in the middle eight, and the third in the least significant eight bits. If there are fewer than three bytes left to encode (or in total), the remaining buffer bits will be zero. The buffer is then used, six bits at a time, most significant first, as indices into the string: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/", and the indicated character is output.

The process is repeated on the remaining data until fewer than four octets remain. If three octets remain, they are processed normally. If fewer than three octets (24 bits) are remaining to encode, the input data is right-padded with zero bits to form an integral multiple of six bits.

After encoding the non-padded data, if two octets of the 24-bit buffer are padded-zeros, two = characters are appended to the output; if one octet of the 24-bit buffer is filled with padded-zeros, one = character is appended. This signals the decoder that the zero bits added due to padding should be excluded from the reconstructed data. This also guarantees that the encoded output length is a multiple of 4 bytes.

PEM requires that all encoded lines consist of exactly 64 printable characters, with the exception of the last line, which may contain fewer printable characters. Lines are delimited by whitespace characters according to local (platform-specific) conventions.

MIME

The MIME (Multipurpose Internet Mail Extensions) specification lists Base64 as one of two binary-to-text encoding schemes (the other being quoted-printable).[5] MIME's Base64 encoding is based on that of the RFC 1421 (https://tools.ietf.org/html/rfc1421) version of PEM: it uses the same 64-character alphabet and encoding mechanism as PEM, and uses the = symbol for output padding in the same way, as described at RFC 2045 (https://tools.ietf.org/html/rfc2045).

MIME does not specify a fixed length for Base64-encoded lines, but it does specify a maximum line length of 76 characters. Additionally it specifies that any extra-alphabetic characters must be ignored by a compliant decoder, although most implementations use a CR/LF newline pair to delimit encoded lines.

Thus, the actual length of MIME-compliant Base64-encoded binary data is usually about 137% of the original data length, though for very short messages the overhead can be much higher due to the overhead of the headers. Very roughly, the final size of Base64-encoded binary data is equal to 1.37 times the original data size + 814 bytes (for headers). The size of the decoded data can be approximated with this formula:

```
bytes = (string_length(encoded_string) - 814) / 1.37
```

UTF-7

UTF-7, described first in RFC 1642 (https://tools.ietf.org/html/rfc1642), which was later superseded by RFC 2152 (https://tools.ietf.org/html/rfc2152), introduced a system called *modified Base64*. This data encoding scheme is used to encode UTF-16 as ASCII characters for use in 7-bit transports such as SMTP. It is a variant of the Base64 encoding used in MIME.[8][9]

The "Modified Base64" alphabet consists of the MIME Base64 alphabet, but does not use the "=" padding character. UTF-7 is intended for use in mail headers (defined in RFC 2047 (https://tools.ietf.org/html/rfc2047)), and the "=" character is reserved in that context as the escape character for "quoted-printable" encoding. Modified Base64 simply omits the padding and ends immediately after the last Base64 digit containing useful bits leaving up to three unused bits in the last Base64 digit.

OpenPGP

OpenPGP, described in RFC 4880 (https://tools.ietf.org/html/rfc4880), describes **Radix-64** encoding, also known as "ASCII armor". Radix-64 is identical to the "Base64" encoding described from MIME, with the addition of an optional 24-bit CRC. The checksum is calculated on the input data before encoding; the checksum is then encoded with the same Base64 algorithm and, prefixed by "=" symbol as separator, appended to the encoded output data.[10]

RFC 3548

RFC 3548 (https://tools.ietf.org/html/rfc3548), entitled *The Base16, Base32, and Base64 Data Encodings*, is an informational (non-normative) memo that attempts to unify the RFC 1421 (https://tools.ietf.org/html/rfc1421) and RFC 2045 (https://tools.ietf.org/html/rfc2045) specifications of Base64 encodings, alternative-alphabet encodings, and the Base32 (which is seldom used) and Base16 encodings.

Unless implementations are written to a specification that refers to RFC 3548 (https://tools.ietf.org/html/rfc3548) and specifically requires otherwise, RFC 3548 forbids implementations from generating messages containing characters outside the encoding alphabet or without padding, and it also declares that decoder implementations must reject data that contain characters outside the encoding alphabet.[6]

RFC 4648

This RFC obsoletes RFC 3548 and focuses on Base64/32/16:

> *This document describes the commonly used Base64, Base32, and Base16 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, use of different encoding alphabets, and canonical encodings.*

URL applications

Base64 encoding can be helpful when fairly lengthy identifying information is used in an HTTP environment. For example, a database persistence framework for Java objects might use Base64 encoding to encode a relatively large unique id (generally 128-bit UUIDs) into a string for use as an HTTP parameter in HTTP forms or HTTP GET URLs. Also, many applications need to encode binary data in a way that is convenient for inclusion in URLs, including in hidden web form fields, and Base64 is a convenient encoding to render them in a compact way.

Using standard Base64 in URL requires encoding of '+', '/' and '=' characters into special percent-encoded hexadecimal sequences ('+' becomes '%2B', '/' becomes '%2F' and '=' becomes '%3D'), which makes the string unnecessarily longer.

For this reason, **modified Base64 for URL** variants exist (such as **base64url** in RFC 4648 (https://tools.ietf.org/html/rfc4648)), where the '+' and '/' characters of standard Base64 are respectively replaced by '-' and '_', so that using URL encoders/decoders is no longer necessary and has no impact on the length of the encoded value, leaving the same encoded form intact for use in relational databases, web forms, and object identifiers in general. Some variants allow or require omitting the padding '=' signs to avoid them being confused with field separators, or require that any such padding be percent-encoded. Some libraries will encode '=' to '.', potentially exposing applications to relative path attacks when a folder name is encoded from user data.

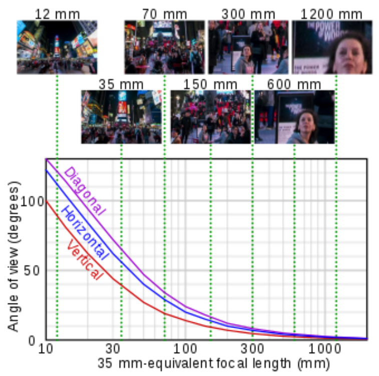
HTML

The `atob()` and `btoa()` JavaScript methods, defined in the HTML5 draft specification,[11] provide Base64 encoding and decoding functionality to web pages. The `btoa()` method outputs padding characters, but these are optional in the input of the `atob()` method.

Other applications

Base64 can be used in a variety of contexts:

- Base64 can be used to transmit and store text that might otherwise cause delimiter collision
- Spammers use Base64 to evade basic anti-spamming tools, which often do not decode Base64 and therefore cannot detect keywords in encoded messages.
- Base64 is used to encode character strings in LDIF files
- Base64 is often used to embed binary data in an XML file, using a syntax similar to `<data encoding="base64">…</data>` e.g. favicons in Firefox's exported `bookmarks.html`.
- Base64 is used to encode binary files such as images within scripts, to avoid depending on external files.
- The data URI scheme can use Base64 to represent file contents. For instance, background images and fonts can be specified in a CSS stylesheet file as `data:` URIs, instead of being supplied in separate files.
- The FreeSWAN IPSec implementation precedes Base64 strings with `0s`, so they can be distinguished from text or hexadecimal strings.



Example of an SVG containing embedded JPEG images encoded in Base64[12]

- Although not part of the official specification for SVG, some viewers can interpret Base64 when used for embedded elements, such as images inside SVG.[13]

Radix-64 applications not compatible with Base64

- Uuencoding, traditionally used on UNIX, uses ASCII 32 (" " (space)) through 95 ("_"), consecutively, making its 64-character set " !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_". Avoiding all lower-case letters was helpful because many older printers only printed uppercase. Using consecutive ASCII characters saved computing power because it was only necessary to add 32, not do a lookup. Its use of most punctuation characters and the space character limits its usefulness.
- BinHex 4 (HQX), which was used within the classic Mac OS, uses a different set of 64 characters. It uses upper and lower case letters, digits, and punctuation characters, but does not use some visually confusable characters like '7', '0', 'g' and 'o'. Its 64-character set is "!"#$%&'()*+,-012345689@ABCDEFGHIJKLMNPQRSTUVXYZ[`abcdefhijklmpqr".
- Several other applications use radix-64 sets more similar to but in a different order to the Base64 format, starting with two symbols, then numerals, then uppercase, then lowercase:
  - Unix stores password hashes computed with crypt in the `/etc/passwd` file using radix-64 encoding called B64. It uses a mostly-alphanumeric set of characters, plus `.` and `/`. Its 64-character set is "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz". Padding is not used.
  - The GEDCOM 5.5 standard for genealogical data interchange encodes multimedia files in its text-line hierarchical file format using radix-64. Its 64-character set is also "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".[14]
  - bcrypt hashes are designed to be used in the same way as traditional crypt(3) hashes, and the algorithm uses a similar but permuted alphabet. Its 64-character set is "./ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".[15]
  - Xxencoding uses a mostly-alphanumeric character set similar to crypt and GEDCOM, but using `+` and `-` rather than `.` and `/`. Its 64-character set is "+-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".
  - 6PACK, used with some terminal node controllers, uses a different set of 64 characters from 0x00 to 0x3f.[16]
  - Bash supports numeric literals in base 2-64, stretching to a character set of `0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ@_`.[17]

See also

- 8BITMIME
- Ascii85 (also called Base85)
- Base16
- Base32
- Base36
- Base62
- Binary-to-text encoding for a comparison of various encoding algorithms
- Binary number
- URL

References

1. "Base64 encoding and decoding - Web APIs | MDN" (https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding).
2. "When to base64 encode images (and when not to)" (https://www.davidbcalhoun.com/2011/when-to-base64-encode-images-and-when-not-to/).
3. *The Base16,Base32,and Base64 Data Encodings* (https://tools.ietf.org/html/rfc4648). IETF. October 2006. doi:10.17487/RFC4648 (https://doi.org/10.17487%2FRFC4648). RFC 4648 (https://tools.ietf.org/html/rfc4648). Retrieved March 18, 2010.
4. *Privacy Enhancement for InternetElectronic Mail: Part I: Message Encryption and Authentication Procedures* (https://tools.ietf.org/html/rfc1421). IETF. February 1993. doi:10.17487/RFC1421 (https://doi.org/10.17487%2FRFC1421). RFC 1421 (https://tools.ietf.org/html/rfc1421). Retrieved March 18, 2010.

5. *Multipurpose Internet Mail Extensions: (MIME) Part One: Format of Internet Message Bodies* (https://tools.ietf.org/html/rfc2045). IETF. November 1996. doi:10.17487/RFC2045 (https://doi.org/10.17487%2FRFC2045). RFC 2045 (https://tools.ietf.org/html/rfc2045). Retrieved March 18, 2010.
6. *The Base16, Base32, and Base64 Data Encodings* (https://tools.ietf.org/html/rfc3548). IETF. July 2003. doi:10.17487/RFC3548 (https://doi.org/10.17487%2FRFC3548). RFC 3548 (https://tools.ietf.org/html/rfc3548). Retrieved March 18, 2010.
7. *Privacy Enhancement for Internet Electronic Mail* (https://tools.ietf.org/html/rfc989). IETF. February 1987. doi:10.17487/RFC0989 (https://doi.org/10.17487%2FRFC0989). RFC 989 (https://tools.ietf.org/html/rfc989). Retrieved March 18, 2010.
8. *UTF-7 A Mail-Safe Transformation Format of Unicode* (https://tools.ietf.org/html/rfc1642). IETF. July 1994. doi:10.17487/RFC1642 (https://doi.org/10.17487%2FRFC1642). RFC 1642 (https://tools.ietf.org/html/rfc1642). Retrieved March 18, 2010.
9. *UTF-7 A Mail-Safe Transformation Format of Unicode* (https://tools.ietf.org/html/rfc2152). IETF. May 1997. doi:10.17487/RFC2152 (https://doi.org/10.17487%2FRFC2152). RFC 2152 (https://tools.ietf.org/html/rfc2152). Retrieved March 18, 2010.
10. *OpenPGP Message Format* (https://tools.ietf.org/html/rfc4880). IETF. November 2007. doi:10.17487/RFC4880 (https://doi.org/10.17487%2FRFC4880). RFC 4880 (https://tools.ietf.org/html/rfc4880). Retrieved March 18, 2010.
11. "7.3. Base64 utility methods" (https://w3c.github.io/html/webappapis.html#atob). *HTML 5.2 Editor's Draft*. World Wide Web Consortium. Retrieved 2 January 2018. Introduced by changeset 5814 (https://html5.org/tools/web-apps-tracker?from=5813&to=5814), 2021-02-01.
12. <image xlink:href="data:image/jpeg;base64,`JPEG contents encoded in Base64`" ... />
13. JSFiddle. "Edit fiddle - JSFiddle" (http://jsfiddle.net/MxHPq/). *jsfiddle.net*.
14. "The GEDCOM Standard Release 5.5" (http://homepages.rootsweb.ancestry.com/~pmcbride/gedcom/55gctoc.htm). Homepages.rootsweb.ancestry.com. Retrieved 2012-06-21.
15. Provos, Niels (1997-02-13). "src/lib/libc/crypt/bcrypt.c r1.1" (https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/crypt/bcrypt.c?rev=1.1&content-type=text/x-cvsweb-markup). Retrieved 2018-05-18.
16. "6PACK a "real time" PC to TNC protocol" (http://private.freepage.de/cgi-bin/feets/freepage_ext/41030x030A/rewrite/alexs/xfr/flexnet/6pack_en/6pack.htm). Retrieved 2013-05-19.
17. "Shell Arithmetic" (https://www.gnu.org/software/bash/manual/html_node/Shell-Arithmetic.html). *Bash Reference Manual*. Retrieved 8 April 2020. "Otherwise, numbers take the form [base#]n, where the optional base is a decimal number between 2 and 64 representing the arithmetic base, and n is a number in that base."

Retrieved from "https://en.wikipedia.org/w/index.php?title=Base64&oldid=1008167594"