What is Base64?

How the binary strings are converted to and from base 64, as well as its uses.

What is Base 64 Encoding?

Introduction

Base64 encoding is a format designed to prevent communication "mishaps" during the transfer of binary information. It achieves this through the conversion of binary data and a "lookup table" — data is eventually made in a stream of *ASCII* characters, which can then be transmitted and decoded. On base 64 encoded data, the resultant string is always larger than the original (i.e. this is *not* a compression algorithm). Another important distinction is that base 64 *does not* encrypt any information — it uses a "standard" table of characters to encode and decode information. In other words, any base-64 string can be decoded, as long as the string was encoded using a standard set of characters (which the decoder can also understand).

Uses for Base64

On top of being used for safely encoding image/media data (you may have seen images on the web encoded in the following format: `(...)`), it is used for SSL certificates, email transmissions, and virtually any transfer of information that requires special (control) characters to be escaped.

Encoding Base64

While many programming languages (e.g. Java, Python, JS) include built-in functions to facilitate the conversion of base64 encoded information and binary data, the algorithm used to perform the conversion is relatively simple. Starting with binary information,

Glossary

Encoding

Any (specific) format used to transfer or save information. This is used in video, file types and more.

Base 64

Base 64 is an encoding format that maps binary values to characters (forming a string).

base 64 splits a binary string into 6-bit groups (note: each zero or one in a binary string is *one bit*) of three bytes. The result is an ASCII-readable string that can be safely transmitted and received.

For the sake of simplicity, we'll use a short string to demonstrate the encoding process. Suppose we have "hi" as our string, the ASCII equivalent being: 104 (**h**), 105 (**i**).

We need to convert the ASCII string to its binary representation:

01101000 01101001 (**0110100001101001** without spaces)

Now, divide the binary representation into 6-bit groups:

011010 000110 100**100**

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	/

With the binary representation of our string, we can then make use of an ASCII lookup table above. We get the following ASCII characters for our string's binary representation:

- 011010 => "a"
- 000110 => "G"
- 100100 => "k"

Before we can finish off the conversion process, it is important to notice the missing padding: base 64 strings *must* be groups of 3 characters (6 byte binary representations). As such, our string "hi" only has 2 characters; this means that we need to insert a "=" at the end of the string to make this a valid base 64 string.

The result is finally: "aGk=" (hi).

Decoding Base64

Decoding base64 strings follows a similar process, just in reverse. Using the result from the previous section ("aGk="), we will use the same Base 64 lookup table:

- "a" => 011010
- "G" => 000110
- "k" => 100100

- "=" => padding

This leaves us with three 6-bit binary groups:

011010 000110 100100

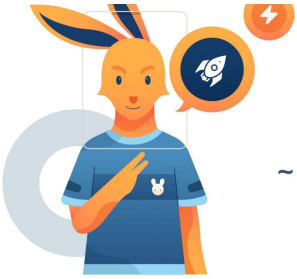
Converting this back to 8-bit groups, we get (the last two zeroes can be ignored):

01101000 01101001

… which results in "hi" in ASCII.

Conclusion

Whether you're sending an email, or trying to encode binary streams (images, videos, etc.), base 64 can be seen in many applications. While the resulting strings are larger, using base 64 encoding is a reliable way to ensure that a transmission of binary information is never "misinterpreted."



~ Making the internet hop faster!