Encoding and Decoding Base64 Strings in Node.js

By Scott Robinson • August 22, 2017 • 6 Comments

What is Base64 Encoding?

Base64 encoding is a way to convert data (typically binary) into the ASCII character set. It is important to mention here that Base64 is not an encryption or compression technique, although it can sometimes be confused as encryption due to the way it seems to obscure data. In fact, size of a Base64 encoded piece of information is 1.3333 times the actual size of your original data.

Base64 is the most widely used base encoding technique with Base16 and Base32 being the other two commonly used encoding schemes.

How Does Base64 Work?

Converting data to base64 is a multistep process. Here is how it works for strings of text:

1. Calculate the 8 bit binary version of the input text
2. Re-group the 8 bit version of the data into multiple chunks of 6 bits
3. Find the decimal version of each of the 6 bit binary chunk
4. Find the Base64 symbol for each of the decimal values via a Base64 lookup table

For a better understanding of this concept, let's take a look at an example.

Suppose we have string "Go win" and we want to convert it into Base64 string. The first step is to convert this string into binary. The binary version of "Go win" is:

```
01000111 01101111 00100000 01110111 01101001 01101110
```

You can see here that each character is represented by 8 bits. However as we said earlier, Base64 converts the data in 8 bit binary form to chunks of 6 bits. This is because Base64 format only has 64 characters: 26 uppercase alphabet letters, 26 lowercase alphabet letters, 10 numeric characters, and the "+" and "/" symbols for new line.

Base64 doesn't use all the ASCII special characters, but only these few. Note that some implementations of Base64 uses different special characters than "+" and "/".

Coming back to the example, let us break our 8 bit data into chunks of 6 bits.

```
010001 110110 111100 100000 011101 110110 100101 101110
```

You won't always be able to divide up the data in to full sets of 6 bits, in which case you'll have to deal with padding.

Now for each chunk above, we have to find its decimal value. These decimal values have been given below:

```
Binary  Decimal
010001  17
110110  54
111100  60
100000  32
011101  29
110110  54
```

```
100101  37
101110  46
```

Finally we have to look the Base64 value for each of the decimal that we just calculated from binary data. Base64 encoding table looks like this:

Base64 Encoding Table

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

Here you can see that decimal 17 corresponds to "R", and decimal 54 corresponds to "2", and so on. Using this encoding table we can see that the string "Go win" is encoded as "R28gd2lu" using Base64. You can use any online text to Base64 converter to verify this result.

Why use Base64 Encoding?

Sending information in binary format can sometimes be risky since not all applications or network systems can handle raw binary. On the other hand, the ASCII character set is widely known and very simple to handle for most systems.

For instance email servers expect textual data, so ASCII is typically used. Therefore, if you want to send images or any other binary file to an email server you first need to encode it in text-based format, preferably ASCII. This is where Base64 encoding comes extremely handy in converting binary data to the correct formats.

Encoding Base64 Strings with Node.js

The easiest way to encode Base64 strings in Node.js is via the Buffer object. In Node.js, `Buffer` is a global object which means that you do not need to use require statement in order to use `Buffer` object in your applications.

Internally `Buffer` is an immutable array of integers that is also capable of performing many different encodings/decodings. These include to/from UTF-8, UCS2, Base64 or even Hex encodings. As you write code that deals with and manipulates data, you'll likely be using the `Buffer` object at some point.

Take a look at the following example. Here we will encode a text string to Base64 using `Buffer` object. Save the following code in a file "encode-text.js"