Why does a base64 encoded string have an = sign at the end

Asked 11 years, 5 months ago Modified 4 months ago Viewed 261k times

445

I know what `base64` encoding is and how to calculate `base64` encoding in C#, however I have seen several times that when I convert a string into base64, there is an `=` at the end.

A few questions came up:

1. Does a `base64` string always end with `=`?
2. Why does an `=` get appended at the end?

encoding base64

Share Follow

edited Nov 5, 2014 at 20:45
Sergey
1,570 1 25 40

asked Aug 2, 2011 at 18:31
santosh singh
27k 26 82 129

1 It kind of seems like this makes it a less effective method of obfuscation in some cases as it is quite detectable. – dgo Feb 25, 2017 at 3:27

20 @user1167442 Base64 is not for obfuscation. It is for transporting binary data (or strings with unicode and other special characters) as a string. – NH. Aug 18, 2017 at 16:24

10 Answers

Sorted by: Highest score (default)

643

Q Does a base64 string always end with `=`?

A: No. (the word `usb` is base64 encoded into `dXNi`)

Q Why does an `=` get appended at the end?

A: As a short answer:
The last character (`=` sign) is added only as a complement (padding) in the final process of encoding a message with a *special* number of characters.

You will not have an `=` sign if your string has a multiple of 3 characters, because `Base64` encoding takes each **three** bytes (a character=1 byte) and represents them as **four** printable

(a) If you want to encode

ABCDEFG <=> [ABC] [DEF] [G]

`Base64` deals with the first block (producing 4 characters) and the second (as they are complete). But for the third, it will add a double `==` in the output in order to complete the 4 needed characters. Thus, the result will be **QUJD REVG Rw==** (without spaces).

[ABC] => QUJD

[DEF] => REVG

[G] => Rw==

(b) If you want to encode **ABCDEFGH** <=> [ABC] [DEF] [GH]

similarly, it will add one `=` at the end of the output to get 4 characters.

The result will be **QUJD REVG R0g=** (without spaces).

[ABC] => QUJD

[DEF] => REVG

[GH] => R0g=

Share Follow

edited May 22, 2022 at 22:43

Alexis Wilke
18.3k 10 79 143

answered Apr 12, 2016 at 10:52

Badr Bellaj
10.5k 2 39 39

320

It serves as padding.

A more complete answer is that a base64 encoded string doesn't *always* end with a `=`, it will only end with one or two `=` if they are required to pad the string out to the proper length.

Share Follow

answered Aug 2, 2011 at 18:33

Andrew Hare
340k 71 636 631

5 "One case in which padding characters are required is concatenating multiple Base64 encoded files." – André Puel Nov 30, 2014 at 19:41

2 @AndréPuel: resynch one single `=` would suffice. If you want to find the boundaries back then a terminator should always be present (and still only one char is needed). The whole padding concept of Base64 is just a brainfart... – 6502 Aug 20, 2015 at 19:07

2 I just wish a relevant and reliable link was posted that explains about padding in `base64` efficiently with illustrations and examples. The present link to wikipedia is absolutely irrelevant like @NH. mentioned. – Fr0zenFyr Jul 5, 2019 at 10:48

4 @Fr0zenFyr If you want a link, en.wikipedia.org/wiki/Base64#Output_padding is pretty good. But the answer by Badr is really a better one (it just hasn't caught up in votes yet). – NH. Jul 9, 2019 at 18:58

77

From Wikipedia:

> The final '==' sequence indicates that the last group contained only one byte, and '=' indicates that it contained two bytes.

Thus, this is some sort of padding.

Share Follow

edited Jan 26, 2018 at 7:23

Jeff
820 9 18

answered Aug 2, 2011 at 18:34

Legolas
1,432 10 11

19

Its defined in RFC 2045 as a special padding character if fewer than 24 bits are available at the end of the encoded data.

Share Follow

edited Nov 19, 2019 at 21:19

answered Aug 2, 2011 at 18:37

iandotkelly
8,964 8 48 67

18

1. No.
2. To pad the Base64-encoded string to a multiple of 4 characters in length, so that it can be decoded correctly.

Share Follow

answered Feb 21, 2014 at 15:28

Ian Kemp
27.4k 18 111 134

5 I removed the = at the end and tested this for 1 million strings. The decoding always matched. – nice_dev Apr 9, 2019 at 11:01

@nice_dev, If the length of the decoded message is known, then the padding is unnecessary. If the length is not known, then there exist circumstances in which the omission of padding could yield invalid results when decoding. For example, in C, the omission of padding could cause the decoding function to instead read uninitialized data (i.e. garbage) and incorrectly interpret that as part of the encoded data. – Tyler Crompton Jun 10, 2022 at 22:58

13

The equals sign (=) is used as padding in certain forms of base64 encoding. The Wikipedia article on base64 has all the details.

Share Follow

answered Aug 2, 2011 at 18:34

Sam Holloway
1,999 15 14

2 Could you explain the logic of why "==" is 1 byte and "=" is 2 bytes? I just can't understand it. How come input: "any carnal pleasure." could get result "YW55IGNhcm5hbCBwbGVhc3VyZS4=", while "any carnal pleasure" could get result "YW55IGNhcm5hbCBwbGVhc3VyZQ==" ? – null Mar 21, 2013 at 6:25

17 It's not that case that '==' is 1 byte and '=' is 2 bytes. It's the case that you need to always have a multiple of 4 bytes in your entire string. So you pad with '=' signs until you get that. The first string has one more character than the second string, so one fewer '=' of padding is required. – Sam Holloway Mar 27, 2013 at 13:31

3 Is this answer supposed to be a comment? – Fr0zenFyr Jul 5, 2019 at 10:57

10

It's padding. From http://en.wikipedia.org/wiki/Base64:

> In theory, the padding character is not needed for decoding, since the number of missing bytes can be calculated from the number of Base64 digits. In some implementations, the padding character is mandatory, while for others it is not used. One case in which

Share Follow

answered Sep 21, 2013 at 8:49

Thomas Leonard
7,008 2 36 40

2 The part about "One case in which padding characters are required is concatenating multiple Base64 encoded files." is wrong. For example when concatenating two base64 files where the source bytes for each file is 3 bytes long the base64 strings will be 4 characters long and have no padding bytes. When you concatenate these two base64 strings there will be no way to tell where one starts and one stops based soley on the concatenated string. So relying on base64 padding to help with that is not going to work. This issue will exist for any file with byte lengths evenly divisible by 3. – RonC Feb 10, 2017 at 14:51

1 I guess it means the case where the final result should be the concatenation of the inputs. e.g. `decode(encode(A)+encode(B))=A+B` works with padding but not without. – Thomas Leonard Feb 11, 2017 at 16:26

perhaps but such limited use doesn't allow the padding char(s) to be relied on for the general case of separating encoded strings when the encoded strings are concatenated together. I only mention it to help developers that may be thinking they can use it that way. – RonC Feb 13, 2017 at 13:52

1 I think your objection really just highlights the difference between the concepts of padding and delimiting. The results of concatenation aren't generally expected to include enough information to make it reversible. You won't know if "c3dpenpsZXJz" was originally "c3dpenps" + "ZXJz" or "c3dp" + "enpsZXJz". But you also don't know if "swizzlers" was originally "swi" + "zzlers" or "swizzl" + "ers". – GargantuChet Apr 21, 2017 at 21:22

1 Copying my comment from a related Base64 padding answer: > Base64 concatenation [with '=' padding] allows encoders to process large chunks in parallel without the burden of aligning the chunk sizes to a multiple of three. Similarly, as an implementation detail, there might be an encoder out there that needs to flush an internal data buffer of a size that is not a multiple of three. – Andre D Sep 5, 2017 at 6:39

http://www.hcidata.info/base64.htm

8

Encoding "Mary had" to Base 64

In this example we are using a simple text string ("Mary had") but the principle holds no matter what the data is (e.g. graphics file). To convert each 24 bits of input data to 32 bits of output, Base 64 encoding splits the 24 bits into 4 chunks of 6 bits. The first problem we notice is that "Mary had" is not a multiple of 3 bytes - it is 8 bytes long. Because of this, the last group of bits is only 4 bits long. To remedy this we add two extra bits of '0' and remember this fact by putting a '=' at the end. If the text string to be converted to Base 64 was 7 bytes long, the last group would have had 2 bits. In this case we would have added four extra bits of '0' and remember this fact by putting '==' at the end.

Share Follow

answered Feb 15, 2015 at 15:52

Dev
151 2 9

4

`=` is a padding character. If the input stream has length that is not a multiple of 3, the padding character will be added. This is required by decoder: if no padding present, the last byte would have an incorrect number of zero bits.

Better and deeper explanation here: https://base64tool.com/detect-whether-provided-string-is-base64-or-not/

Share Follow

answered Jun 24, 2020 at 18:50

Vladimir Ignatyev
1,992 18 33

1 To expand on this, while standard base64 specifies padding, it is not because it can't be decoded without it. It is possible to make a base64 implementation whose decoder does not require padding, and the decoder can still obtain all the same information from the position of the end of the string. Padding allows the following extra benefits: 1) that base64 strings will all be a multiple of 4 characters long, which may simplify decoder design, and 2) that you can concatenate two base64 strings without re-encoding and there is enough information at the break to properly get back into sync. – thomasrutter Apr 23, 2021 at 8:08

This is not true for JavaScript (and maybe other languages). When you call `btoa('ipsum')` in your console you get `aXBzdW0=`. But removing the = sign and decoding `atob('aXBzdW0')` still results in `ipsum`. Maybe `atob` pads internally? – Sjeiti Jun 3, 2022 at 9:08

@Sjeiti if you looked into my implementation of base64 encoding and decoding, you could see that padding is an optional thing. Check out github.com/vladignatyev/base64tool/blob/master/modules/base64/... – Vladimir Ignatyev Jun 4, 2022 at 10:20

Ok, I was referring to the sentence "This is required by decoder (...)" which makes it seem mandatory. – Sjeiti Jun 14, 2022 at 6:28

1

The equals or double equals serves as padding. It's a stupid concept defined in RFC2045 and it is actually superfluous. Any decend parser can encode and decode a base64 string without knowing about padding by just counting up the number of characters and filling in the rest if size isn't dividable by 3 or 4 respectively. This actually leads to difficulties every now and then, because some parsers expect padding while others blatantly ignore it. My MPU base64 decoder for example needs padding, but it receives a non-padded base64 string over the network. This leads to erronous parsing and I had to account for it myself.

Share Follow

answered Aug 29, 2022 at 16:48

glades
2,485 9 25