Why do we use Base64?

Asked 9 years, 6 months ago Active 2 months ago Viewed 84k times

260

122

Wikipedia says

> Base64 encoding schemes are commonly used when there is a need to encode binary data that needs be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport.

But is it not that data is always stored/transmitted in binary because the memory that our machines have store binary and it just depends how you interpret it? So, whether you encode the bit pattern `010011010110000101101110` as `Man` in ASCII or as `TWFu` in Base64, you are eventually going to store the same bit pattern.

If the ultimate encoding is in terms of zeros and ones and every machine and media can deal with them, how does it matter if the data is represented as ASCII or Base64?

What does it mean "media that are designed to deal with textual data"? They can deal with binary => they can deal with anything.

Thanks everyone, I think I understand now.

When we send over data, we cannot be sure that the data would be interpreted in the same format as we intended it to be. So, we send over data coded in some format (like Base64) that both parties understand. That way even if sender and receiver interpret same things differently, but because they agree on the coded format, the data will not get interpreted wrongly.

From Mark Byers example

If I want to send

```
Hello
world!
```

One way is to send it in ASCII like

```
72 101 108 108 111 10 119 111 114 108 100 33
```

But byte 10 might not be interpreted correctly as a newline at the other end. So, we use a subset of ASCII to encode it like this

```
83 71 86 115 98 71 56 115 67 110 100 118 99 109 120 107 73 61 61
```

which at the cost of more data transferred for the same amount of information ensures that the receiver can decode the data in the intended way, even if the receiver happens to have different interpretations for the rest of the character set.

algorithm character-encoding binary ascii base64

edited Sep 11 '17 at 10:02

mega6382
8,094 15 38 60

asked Aug 21 '10 at 15:21

Lazer
70.9k 101 251 338

6 Historical background: Email servers used to be 7-bit ASCII. Many of them would set the high bit to 0 so you had to send 7-bit values only. See en.wikipedia.org/wiki/Email#Content_encoding – Harold L Aug 21 '10 at 15:40

48 We use base64 because it's more readable than Perl – Martin Aug 21 '10 at 16:28

2 @Martin, you are kidding. Perl is hard to read, but base64 is unreadable at all. – Peter Long Jul 11 '11 at 7:40

1 @Lazer Your image is missing – Mick Sep 28 '15 at 3:44

2 @Lazer, "But byte 10 might not be interpreted correctly as a newline at the other end." why? the two parties have agreed upon ASCII and they must be interpreting it correctly! – ProgramCpp Aug 24 '17 at 18:01

12 Answers

279

Your first mistake is thinking that ASCII encoding and Base64 encoding are interchangeable. They are not. They are used for different purposes.

- When you encode text in ASCII, you start with a text string and convert it to a sequence of bytes.
- When you encode data in Base64, you start with a sequence of bytes and convert it to a text string.

To understand why Base64 was necessary in the first place we need a little history of computing.

Computers communicate in binary - 0s and 1s - but people typically want to communicate with more rich forms data such as text or images. In order to transfer this data between computers it first has to be encoded into 0s and 1s, sent, then decoded again. To take text as an example - there are many different ways to perform this encoding. It would be much simpler if we could all agree on a single encoding, but sadly this is not the case.

Originally a lot of different encodings were created (e.g. Baudot code) which used a different number of bits per character until eventually ASCII became a standard with 7 bits per character. However most computers store binary data in bytes consisting of 8 bits each so ASCII is unsuitable for tranferring this type of data. Some systems would even wipe the most significant bit.

Furthermore the difference in line ending encodings across systems mean that the ASCII character 10 and 13 were also sometimes modified.

To solve these problems Base64 encoding was introduced. This allows you to encode aribtrary bytes to bytes which are known to be safe to send without getting corrupted (ASCII alphanumeric characters and a couple of symbols). The disadvantage is that encoding the message using Base64 increases its length - every 3 bytes of data is encoded to 4 ASCII characters.

To send text reliably you can **first** encode to bytes using a text encoding of your choice (for example UTF-8) and then **afterwards** Base64 encode the resulting binary data into a text string that is safe to send encoded as ASCII. The receiver will have to reverse this process to recover the original message. This of course requires that the receiver knows which encodings were used, and this information often needs to be sent separately.

Historically it has been used to encode binary data in email messages where the email server might modify line-endings. A more modern example is the use of Base64 encoding to embed image data directly in HTML source code. Here it is necessary to encode the data to avoid characters like '<' and '>' being interpreted as tags.

Here is a working example:

I wish to send a text message with two lines:

```
Hello
world!
```

If I send it as ASCII (or UTF-8) it will look like this:

```
72 101 108 108 111 10 119 111 114 108 100 33
```

The byte 10 is corrupted in some systems so we can base 64 encode these bytes as a Base64 string:

```
SGVsbG8sCndvcmxkIQ==
```

Which when encoded using ASCII looks like this:

```
83 71 86 115 98 71 56 115 67 110 100 118 99 109 120 107 73 61 61
```

All the bytes here are known safe bytes, so there is very little chance that any system will corrupt this message. I can send this instead of my original message and let the receiver reverse the process to recover the original message.

edited Nov 30 '19 at 8:55

FlyingSixtySix
51 1 8

answered Aug 21 '10 at 15:44

Mark Byers
649k 146 1433 1368

4 "most modern communications protocols won't corrupt data" - although for example email might, with a

delivery agent replacing the string of characters "\nFrom " with "\n>From " when it saves the message to a mailbox. Or HTTP headers are newline terminated with no reversible way to escape newlines in the data (line continuation conflates whitespace), so you can't just dump arbitrary ASCII into them either. base64 is better than *just* 7-bit safe, it's alpha-numeric-and-=+/ safe. – Steve Jessop Aug 21 '10 at 16:12

1 "The disadvantage is that encoding the message using Base64 increases its length - every 3 bytes of data is encoded to 4 bytes." How does it increase to 4 bytes? Won't it will still be 3*8 = 24 bits only? – Lazer Aug 21 '10 at 16:15

4 @Lazer: no. Look at your own example - "Man" is base-64 encoded as "TWFu". 3 bytes -> 4 bytes. It's because the input is allowed to be any of the 2^8 = 256 possible bytes, whereas the output only uses 2^6 = 64 of them (and =, to help indicate the length of the data). 8 bits per quartet of output are "wasted", in order to prevent the output from containing any "exciting" characters even though the input does. – Steve Jessop Aug 21 '10 at 16:20

@Steve Jessop: For "TWFu", each character takes only 6 bits each. That makes it 3 bytes. What am I missing? – Lazer Aug 21 '10 at 16:57

@Lazer: I have added a worked example to my answer. Does this help clarify things? – Mark Byers Aug 21 '10 at 17:16

Very good answer. Thank you Mark! I am just wondering, though, why is the last conversion (from Base64 to a *new* ASCII) necessary? Couldn't we just send that Base64 and then the receiver will decode it to the original ASCII? Why is the new conversion to a new ASCII necessary before sending the message? – Tiago Jul 24 '12 at 2:42

Computers communicate using bits and bytes. When you send a text message from one computer to another it has to be converted to and from bytes. This is done using a character encoding. If you don't do this explicitly, then it is probably being done for you automatically using the OS or application's default text encoding. Of course this can be convenient, but it's usually best to explicitly encode your data with a specific encoding rather than relying on the defaults as the defaults could change or be different on different computers. – Mark Byers Jul 24 '12 at 21:38

It might be helpful to restate "When you encode data in Base64, you start with a sequence of bytes and convert it to a text string" as "When you encode data in Base64, you start with a sequence of bytes and convert it to a sequence of bytes consisting only of ASCII values". A sequence of bytes consisting only of ASCII characters is what's required by SMTP, which is why Base64 (and quoted-printable) are used as content-transfer-encodings. Excellent overview! – ALEXintlsos Mar 28 '13 at 19:20

1 I would vote, but has 64 votes. Sorry this is perfect. – Jessé Catrinck Jan 7 '16 at 18:31

“The byte 10 is corrupted in some systems” - hey @Mark Byers can I ask how would you know this? I mean there must be some proof behind this, right? – Joshua Jun 5 '17 at 7:19

I find an back refered post talking about this "If we don't do this, then there is a risk that certain characters may be improperly interpreted. For e.g. Newline chars such as 0x0A and 0x0D, Control characters like ^C, ^D, and ^Z that are interpreted as end-of-file on some platforms, NULL byte as the end of a text string, Bytes above 0x7F (non-ASCII), We use Base64 encoding in HTML/XML docs to avoid characters like '<' and '>' being interpreted as tags." – Joshua Jun 5 '17 at 7:26

Encoding binary data in XML

59

Suppose you want to embed a couple images within an XML document. The images are binary data, while the XML document is text. But XML cannot handle embedded binary data. So how do you do it?

One option is to encode the images in base64, turning the binary data into text that XML can handle.

Instead of:

```
<images>
  <image name="Sally">{binary gibberish that breaks XML parsers}</image>
  <image name="Bobby">{binary gibberish that breaks XML parsers}</image>
</images>
```

you do:

```
<images>
  <image name="Sally" encoding="base64">j23894uaiAJSD3234kljasjkSD...</image>
  <image name="Bobby" encoding="base64">Ja3k23JKasil3452AsdfjlksKsasKD...</image>
</images>
```

And the XML parser will be able to parse the XML document correctly and extract the image data.

edited Jun 19 '18 at 21:24
Sridhar Sarnobat
15.6k 8 66 72

answered Aug 21 '10 at 15:37
yfeldblum
61.1k 9 124 163

3 Best answer here. – Alan Smith Mar 31 '17 at 13:49

This might be how Microsoft's old `.mht` format works (html file + images in a single file). – Sridhar Sarnobat Jun 19 '18 at 21:24

37

Why not look to the RFC that currently defines Base64?

> Base encoding of data is used in many situations to store or transfer data in environments that, perhaps for legacy reasons, are restricted to US-ASCII [1] 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. Multipurpose Internet Mail Extensions (MIME) [4] 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.

Base64 was originally devised as a way to allow binary data to be attached to emails as a part of the Multipurpose Internet Mail Extensions.

answered Aug 21 '10 at 15:39
Billy ONeal
92.7k 39 274 508

26

Media that is designed for textual data is of course eventually binary as well, but textual media often use certain binary values for control characters. Also, textual media may reject certain binary values as non-text.

Base64 encoding encodes binary data as values that can only be interpreted as text in textual media, and is free of any special characters and/or control characters, so that the data will be preserved across textual media as well.

answered Aug 21 '10 at 15:25

Håvard S

19.7k 4 54 66

So its like with Base64, mostly both the source and destination will interpret the data the same way, because most probably they will interpret these 64 characters the same way, even if they interpret the control characters in different ways. Is that right? – Lazer Aug 21 '10 at 16:17

6 They data may even be destroyed in transit. For example many FTP programs rewrite line endings from 13,10 to 10 or via versa if the operating system of the server and client don't match and the transfer is flagged as text mode. FTP is just the first example that came to my mind, it is not a good one because FTP does support a binary mode. – Hendrik Brummermann Aug 21 '10 at 17:13

@nhnb: I think FTP is a fine example since it shows that text-mode is unsuitable for things that want binary data. – jamesdlin Aug 21 '10 at 17:22

What is a textual media? – Koray Tugay Mar 29 '17 at 10:02

17

It is more that the media **validates** the string encoding, so we want to ensure that the data is acceptable by a handling application (and doesn't contain a binary sequence representing EOL for example)

Imagine you want to send binary data in an email with encoding UTF-8 -- The email may not display correctly if the stream of ones and zeros creates a **sequence** which isn't valid Unicode in UTF-8 encoding.

The same type of thing happens in URLs when we want to encode characters not valid for a URL in the URL itself:

> http://www.foo.com/hello my friend -> http://www.foo.com/hello%20my%20friend

This is because we want to send a space over a system that will think the space is smelly.

All we are doing is ensuring there is a 1-to-1 mapping between a known good, acceptable and non-detrimental sequence of bits to another literal sequence of bits, and that the handling application **doesn't distinguish** the encoding.

In your example, `man` may be valid ASCII in first form; but often you may want to transmit values that are random binary (ie sending an image in an email):

> MIME-Version: 1.0
> Content-Description: "Base64 encode of a.gif"
> Content-Type: image/gif; name="a.gif"
> Content-Transfer-Encoding: Base64
> Content-Disposition: attachment; filename="a.gif"

Here we see that a GIF image is encoded in base64 as a chunk of an email. The email client reads the headers and decodes it. Because of the encoding, we can be sure the GIF doesn't contain anything that may be interpreted as protocol and we avoid inserting data that SMTP or POP may find significant.

edited Aug 21 '10 at 15:35

answered Aug 21 '10 at 15:25

Aiden Bell
26.5k 3 65 114

1 That's awesome--this explanation made it click. It's not to obfuscate or compress data, but simply to avoid using special sequences that can be interpreted as protocol. – Patrick Michaelsen Sep 10 '19 at 23:13

12

One example of when I found it convenient was when trying to embed binary data in XML. Some of the binary data was being misinterpreted by the SAX parser because that data could be literally anything, including XML special characters. Base64 encoding the data on the transmitting end and decoding it on the receiving end fixed that problem.

edited May 23 '17 at 12:34

Community ♦
1 1

answered Aug 21 '10 at 15:33

Bill the Lizard
346k 164 523 812

1 +1 -- but this is by no means SAX specific. It would happen to any XML parser, i.e. DOM or XLINQ. – Billy ONeal Aug 21 '10 at 15:35

1 @Billy: Yes, absolutely. I just happened to be using a SAX parser for that application. – Bill the Lizard Aug 21 '10 at 15:36

Different engines, for example the SAX parser might interpret some of the ASCII values in different ways (different control characters). So, the idea here is to use the subset of ASCII that has the common meaning universally. Right? – Lazer Aug 21 '10 at 16:21

1 @Lazer: Right. Unencoded binary data will have control characters in it just by chance when you try to interpret it as ASCII (which in this case it was not). – Bill the Lizard Aug 21 '10 at 17:16

11

Base64 instead of escaping special characters

I'll give you a very different but real example: I write javascript code to be run in a browser. HTML tags have ID values, but there are constraints on what characters are valid in an ID.

But I want my ID to losslessly refer to files in my file system. Files in reality can have all manner of weird and wonderful characters in them from exclamation marks, accented characters, tilde, even

emoji! I cannot do this:

```
<div id="/path/to/my_strangely_named_file!@().jpg">
    <img src="http://myserver.com/path/to/my_strangely_named_file!@().jpg">
    Here's a pic I took in Moscow.
</div>
```

Suppose I want to run some code like this:

```
# ERROR
document.getElementById("/path/to/my_strangely_named_file!@().jpg");
```

I think this code will fail when executed.

With Base64 I can refer to something complicated without worrying about which language allows what special characters and which need escaping:

```
document.getElementById("18GerPD8fY4iTbNpC9hHNXNHyrDMampPLA");
```

Unlike using an MD5 or some other hashing function, you can reverse the encoding to find out what exactly the data was that actually useful.

I wish I knew about Base64 years ago. I would have avoided tearing my hair out with ‘ `encodeURIComponent` ’ and `str.replace('\n','\\n')`

SSH transfer of text:

If you're trying to pass complex data over ssh (e.g. a dotfile so you can get your shell personalizations), good luck doing it without Base 64. This is how you would do it with base 64 (I know you can use SCP, but that would take multiple commands - which complicates key bindings for sshing into a server):

- https://superuser.com/a/1376076/114723

edited Jun 29 '19 at 23:32

answered Sep 8 '17 at 22:34

Sridhar Sarnobat
15.6k 8 66 72

10

Most computers store data in 8-bit binary format, but this is not a requirement. Some machines and transmission media can only handle 7 bits (or maybe even lesser) at a time. Such a medium would interpret the stream in multiples of 7 bits, so if you were to send 8-bit data, you won't receive what you expect on the other side. Base-64 is just one way to solve this problem: you encode the input into a 6-bit format, send it over your medium and decode it back to 8-bit format at the receiving end.

answered Aug 21 '10 at 15:32

casablanca
62.9k 5 117 140

3 Why is it a problem if stream interrupts after 7 bits. At the end, the other machine will have all the data

received over the stream, it can then choose 8 bits format for displaying it? What's wrong with my mind! – mallaudin May 14 '17 at 19:07

Why/ How do we use Base64 encoding?

5

Base64 is one of the binary-to-text encoding scheme having 75% efficiency. It is used so that typical binary data (such as images) may be safely sent over legacy "not 8-bit clean" channels. In earlier email networks (till early 1990s), most email messages were plain text in the 7-bit US-ASCII character set. So many early comm protocol standards were designed to work over "7-bit" comm links "not 8-bit clean". Scheme efficiency is the ratio between number of bits in the input and the number of bits in the encoded output. Hexadecimal (Base16) is also one of the binary-to-text encoding scheme with 50% efficiency.

Base64 Encoding Steps (Simplified):

1. Binary data is arranged in continuous chunks of 24 bits (3 bytes) each.
2. Each 24 bits chunk is grouped in to four parts of 6 bit each.
3. Each 6 bit group is converted into their corresponding Base64 character values, i.e. Base64 encoding converts three octets into four encoded characters. The ratio of output bytes to input bytes is 4:3 (33% overhead).
4. Interestingly, the same characters will be encoded differently depending on their position within the three-octet group which is encoded to produce the four characters.
5. The receiver will have to reverse this process to recover the original message.

answered Apr 17 '18 at 19:01

Mushtaq Hussain
661 6 5

5

> What does it mean "media that are designed to deal with textual data"?

That those protocols were designed to handle text (often, only *English* text) instead of binary data (like .png and .jpg images).

> They can deal with binary => they can deal with anything.

But the converse is not true. A protocol designed to represent text may improperly treat binary data that happens to contain:

- The bytes 0x0A and 0x0D, used for line endings, which differ by platform.
- Other control characters like 0x00 (NULL = C string terminator), 0x03 (END OF TEXT), 0x04 (END OF TRANSMISSION), or 0x1A (DOS end-of-file) which may prematurely signal the end of data.
- Bytes above 0x7F (if the protocol that was designed for ASCII).
- Byte sequences that are invalid UTF-8.

So you can't just send binary data over a text-based protocol. You're limited to the bytes that represent the non-space non-control ASCII characters, of which there are 94. The reason Base 64 was chosen was that it's faster to work with powers of two, and 64 is the largest one that works.

> One question though. How is that systems still don't agree on a common encoding technique like the so common UTF-8?

On the Web, at least, they mostly have. A majority of sites use UTF-8.

The problem in the West is that there is a lot of old software that ass-u-me-s that 1 byte = 1 character and can't work with UTF-8.

The problem in the East is their attachment to encodings like GB2312 and Shift_JIS.

And the fact that Microsoft seems to have still not gotten over having picked the wrong UTF encoding. If you want to use the Windows API or the Microsoft C runtime library, you're limited to UTF-16 or the locale's "ANSI" encoding. This makes it painful to use UTF-8 because you have to convert all the time.

answered Aug 21 '10 at 18:24

dan04
70.1k 20 144 178

5

In addition to the other (somewhat lengthy) answers: even ignoring old systems that support only 7-bit ASCII, basic problems with supplying binary data in text-mode are:

- Newlines are typically transformed in text-mode.
- One must be careful not to treat a NUL byte as the end of a text string, which is all too easy to do in any program with C lineage.

answered Aug 21 '10 at 17:27

jamesdlin
33k 6 76 107

There are also control characters like ^C, ^D, and ^Z which get interpreted as end-of-file on some platforms. – dan04 Aug 21 '10 at 17:35

3

> What does it mean "media that are designed to deal with textual data"?

Back in the day when ASCII ruled the world dealing with non-ASCII values was a headache. People jumped through all sorts of hoops to get these transferred over the wire without losing out information.

answered Aug 21 '10 at 15:24

dirkgently
95.8k 16 118 179

3 Actually, back in the day, ASCII wasn't even used everywhere. Many protocols had a separate text-mode and binary-mode for transferring data, unfortunately email didn't back then. Text-mode is necessary precisely because no single text encoding ruled the world, not ASCII; every computer network has their own favourite encoding, so there are gateways whose job is to convert the exchanged text to the local encoding so that a Japanese company can send email to an American business consultant without mojibake. This conversion, obviously, is undesirable when sending binary data. – Lie Ryan Dec 10 '14 at 9:00