# How does UTF-8 encoding identify single byte and double byte characters?

Asked 7 years, 9 months ago Modified 4 years, 6 months ago Viewed 30k times

41

Recently I've faced an issue regarding character encoding, while I was digging into character set and character encoding this doubt came to my mind.UTF-8 encoding is most popular because of its backward compatibility with ASCII.Since UTF-8 is variable length encoding format, how it differentiates single byte and double byte characters.For example, "Aني" is stored as "410754" (Unicode for A is 41 and Unicode for Arabic character is 0754.How encoding identifies 41 is one character and 0754 is another two-byte character?Why it's not considered as 4107 as one double byte character and 54 as a single byte character?

unicode encoding utf-8 character-encoding

Share Improve this question Follow

edited Jun 15, 2017 at 19:51

asked Jun 15, 2017 at 11:03



Ganesh kumar S R

601 ● 1 ● 8 ● 10

3 FYI, "non-english" makes little sense, since even in English many "foreign" characters are regularly used; you're → naïve ← if you think otherwise. ;) – deceze ♦ Jun 15, 2017 at 12:59

There is no strong concept of "double byte" characters in UTF-8. UTF-8 encodes each Unicode codepoint in one to four code units. There is nothing special about two vs three. – Tom Blodget Jul 26, 2019 at 23:54

Actually "Aني" stored as "00410754", 2 16-bit characters. The string UTF-8 encoded as pointed out below is "41DD94". Your post is a little confusing as it could be understood as saying that the UTF-8 is "410754". – crass Sep 10, 2021 at 6:10

If you want to determine how many bytes a particular character requires, `var byteCount = Encoding.UTF8.GetByteCount(new char[] {ch});` – Eric J. Dec 23, 2021 at 0:08

above one is c# code. Here's Javascript code if anyone requires (new TextEncoder().encode("ني")).length – Anmol Saraf May 11, 2022 at 3:18

## 3 Answers

Sorted by: Highest score (default)

**83**

> For example, "Aبـ" is stored as "410754"

That's not how UTF-8 works.

Characters U+0000 through U+007F (aka ASCII) are stored as single bytes. They are the only characters whose codepoints numerically match their UTF-8 presentation. For example, U+0041 becomes `0x41` which is `01000001` in binary.

All other characters are represented with multiple bytes. U+0080 through U+07FF use two bytes each, U+0800 through U+FFFF use three bytes each, and U+10000 through U+10FFFF use four bytes each.

Computers know where one character ends and the next one starts because UTF-8 was designed so that the single-byte values used for ASCII do not overlap with those used in multi-byte sequences. The bytes `0x00` through `0x7F` are only used for ASCII and nothing else; the bytes above `0x7F` are only used for multi-byte sequences and nothing else. Furthermore, the bytes that are used at the beginning of the multi-byte sequences also cannot occur in any other position in those sequences.

Because of that the codepoints need to be encoded. Consider the following binary patterns:

- 2 bytes: `110xxxxx 10xxxxxx`
- 3 bytes: `1110xxxx 10xxxxxx 10xxxxxx`
- 4 bytes: `11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`

The amount of ones in the first byte tells you how many of the following bytes still belong to the same character. All bytes that belong to the sequence start with `10` in binary. To encode the character you convert its codepoint to binary and fill in the x's.

As an example: U+0754 is between U+0080 and U+07FF, so it needs two bytes. `0x0754` in binary is `11101010100`, so you replace the x's with those digits:

110**11101** 10**010100**

Share Improve this answer Follow

edited Sep 21, 2020 at 9:55

answered Jun 15, 2017 at 12:56

CharlotteBuff
4,469 ● 1 ● 21 ● 21

Now I understand how UTF-8 encoding works and how it identifies the single byte and double characters from the sequence of bytes.So UTF-8 follows this pattern to identify the contiguous bytes of a single character, So the other encoding format also has this kind of pattern to separate the bytes for the single character right? – Ganesh kumar S R Jun 15, 2017 at 19:35

5 The other two Unicode formats are UTF-16 and UTF-32. UTF-32 uses units of four bytes each which is more than enough for all possible Unicode values, so all codepoints are simply saved unmodified. U+0754 becomes 00 00 07 54.UTF-16 uses units of two bytes each which is enough for all characters up to U+FFFF, so U+0754 becomes 07 54. Everything beyond FFFF is encoded with two so-called surrogates, which are special two-byte codepoints that aren't used by any characters. Again, codepoints for starting surrogates and ending surrogates don't overlap. – CharlotteBuff Jun 15, 2017 at 20:49

See the detailed explanations and examples on Wikipedia: UTF-8, UTF-16 – Remy Lebeau Jun 16, 2017 at 4:18

@RandomGuy32 Thanks for the answer. I'm also trying to get my head around this stuff. "U+0754 is between U+0080 and U+07FF" , I don't quite get this number system. How can you tell it's between? Also, is this "0x0754" consider hexidemical format? I don't get how you were able to convert it binary format. Thank you. – Moondra Feb 10, 2018 at 23:05

@Moondra Yes, Unicode code points are most commonly referenced in hexadecimal, so U+0754 simply means the hexadecimal number 0754, which is 1876 in decimal. – CharlotteBuff Feb 11, 2018 at 1:11

36

**Short answer:**

UTF-8 is designed to be able to **unambiguously** identify the **type** of each **byte** in a text stream:

- **1-byte codes** (all and only the ASCII characters) start with a **0**
- **Leading bytes of 2-byte codes** start with two 1s followed by a 0 (i.e. **110**)
- **Leading bytes of 3-byte codes** start with three 1s followed by a 0 (i.e. **1110**)
- **Leading bytes of 4-byte codes** start with four 1s followed by a 0 (i.e. **11110**)
- **Continuation bytes** (of all multi-byte codes) start with a single 1 followed by a 0 (i.e. **10**)

Your example Aپ, which consists of the Unicode code points U+0041 and U+0754, is encoded in UTF-8 as:

> **0**1000001 **110**11101 **10**010100

So, when decoding, UTF-8 knows that the first byte must be a 1-byte code, the second byte must be the leading byte of a 2-byte code, the third byte must be a continuation byte, and since the second byte is the leading byte of a *2-byte* code, the second and third byte together must form this 2-byte code.

---

See here how UTF-8 encodes Unicode code points.

Share Improve this answer Follow

edited May 28, 2020 at 21:34

Brian Donovan

**8,410** ● 1 ● 28 ● 25

answered Jun 27, 2017 at 9:10

weibeld

**15.4k** ● 2 ● 39 ● 57

---

**1**

Just to clarify, ASCII mean standard 7-bit ASCII and not extended 8-bit ASCII as commonly used in Europe.

Thus, part of first byte (0x80 to 0xFF) goes to dual byte representation and part of second byte on two bytes (0x0800 to 0xFFFF) takes the full three-byte representation.

Four byte representation uses only the lowest three bytes and only 1.114.111 of the 16.777.215 available possibilities

You have an xls here

That means that interpreters must 'jump back' a NUL (0) byte when they find those binary patterns.

Hope this helps somebody!

Share Improve this answer Follow

answered Jul 26, 2019 at 16:34

jmcollantes

**96** ● 1 ● 9

**Start asking to get answers**

**Explore related questions**

Find the answer to your question by asking.

Ask question

unicode encoding utf-8 character-encoding

See similar questions with these tags.