What's the difference between UTF-8 and UTF-8 with BOM?

Asked 15 years, 1 month ago Modified 1 year, 5 months ago Viewed 992k times

What's different between UTF-8 and UTF-8 with BOM?

1163

unicode utf-8 character-encoding byte-order-mark

Share Follow

edited Sep 20, 2023 at 20:20
TylerH
21.1k ● 77 ● 79 ● 112

asked Feb 8, 2010 at 18:26
simple
11.8k ● 3 ● 19 ● 11

91 UTF-8 can be auto-detected better by contents than by BOM. The method is simple: try to read the file (or a string) as UTF-8 and if that succeeds, assume that the data is UTF-8. Otherwise assume that it is CP1252 (or some other 8 bit encoding). Any non-UTF-8 eight bit encoding will almost certainly contain sequences that are not permitted by UTF-8. Pure ASCII (7 bit) gets interpreted as UTF-8, but the result is correct that way too. – Tronic Feb 11, 2010 at 13:25

54 Scanning large files for UTF-8 content takes time. A BOM makes this process much faster. In practice you often need to do both. The culprit nowadays is that still a lot of text content isn't Unicode, and I still bump into tools that say they do Unicode (for instance UTF-8) but emit their content a different codepage. – Jeroen Wiert Pluimers Dec 18, 2013 at 7:41

12 @Tronic I don't really think that "*better*" fits in this case. It depends on the environment. If you are **sure** that all UTF-8 files are marked with a **BOM** than checking the **BOM** is the "*better*" way, because it is faster and more reliable. – mg30rg Jul 31, 2014 at 9:31

41 UTF-8 does not have a BOM. When you put a U+FEFF code point at the start of a UTF-8 file, special care must be made to deal with it. This is just one of those Microsoft naming lies, like calling an encoding "Unicode" when there is no such thing. – tchrist Oct 1, 2014 at 22:37

10 "*The modern Mainframe (and AIX) is little endian UTF-8 aware*" UTF-8 doesn't have an **endedness**! there is no shuffling of bytes around to put pairs or groups of four into the right "order" for a particular system! To detect a UTF-8 byte sequence it may be useful to note that the first byte of a multi-byte sequence "codepoint" (the bytes that are NOT "plain" ASCII ones) has the MS bit set and all one to three more successively less significant bits followed by a reset bit. The total number of those set bits is one less bytes that are in that codepoint and they will ALL have the MSB set... – SlySven Aug 19, 2016 at 17:38

22 Answers

Sorted by: Highest score (default)

The UTF-8 BOM is a sequence of *bytes* at the start of a text stream (`0xEF, 0xBB, 0xBF`) that

1038

allows the reader to more reliably guess a file as being encoded in UTF-8.

Normally, the BOM is used to signal the endianness of an encoding, but since endianness is irrelevant to UTF-8, the BOM is unnecessary.

According to the Unicode standard, the **BOM for UTF-8 files is not recommended**:

> ### 2.6 Encoding Schemes
>
> ... Use of a BOM is neither required nor recommended for UTF-8, but may be encountered in contexts where UTF-8 data is converted from other encoding forms that use a BOM or where the BOM is used as a UTF-8 signature. See the "Byte Order Mark" subsection in *Section 16.8, Specials*, for more information.

Share Follow

edited Apr 16, 2020 at 22:43

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Feb 8, 2010 at 18:33

Martin Cote
29.9k ● 15 ● 77 ● 99

156 It might not be recommended but from my experience in Hebrew conversions the BOM is sometimes crucial for UTF-8 recognition in Excel, and may make the difference between Jibrish and Hebrew – Matanya Dec 7, 2012 at 8:13

54 It might not be recommended but it did wonders to my powershell script when trying to output "æøå" – Marius Nov 12, 2013 at 9:22

84 Regardless of it not being recommended by the standard, it's allowed, and I greatly prefer having something to act as a UTF-8 signature rather the alternatives of assuming or guessing. Unicode-compliant software should/must be able to deal with its presence, so I personally encourage its use. – martineau Dec 31, 2013 at 20:41

34 @bames53: Yes, in an ideal world storing the encoding of text files as file system metadata would be a better way to preserve it. But most of us living in the real world can't change the file system of the OS(s) our programs get run on -- so using the Unicode standard's platform-independent BOM signature seems like the best and most practical alternative IMHO. – martineau Jan 16, 2014 at 19:37

48 @martineau Just yesterday I ran into a file with a UTF-8 BOM that wasn't UTF-8 (it was CP936). What's unfortunate is that the ones responsible for the immense amount of pain cause by the UTF-8 BOM are largely oblivious to it. – bames53 Jan 16, 2014 at 23:21

308

The other excellent answers already answered that:

- There is no official difference between UTF-8 and BOM-ed UTF-8
- A BOM-ed UTF-8 string will start with the three following bytes. `EF BB BF`
- Those bytes, if present, must be ignored when extracting the string from the file/stream.

But, as additional information to this, the BOM for UTF-8 could be a good way to "smell" if a string was encoded in UTF-8... Or it could be a legitimate string in any other encoding...

For example, the data [EF BB BF 41 42 43] could either be:

- The legitimate ISO-8859-1 string "ï»¿ABC"
- The legitimate UTF-8 string "ABC"

So while it can be cool to recognize the encoding of a file content by looking at the first bytes, you should not rely on this, as show by the example above

Encodings should be known, not divined.

Share Follow

edited May 6, 2015 at 19:25

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Feb 8, 2010 at 18:42

paercebal
83.4k ● 38 ● 134 ● 160

74 @Alcott : You understood correctly. The string [EF BB BF 41 42 43] is just a bunch of bytes. You need external information to choose how to interpret it. If you believe those bytes were encoded using ISO-8859-1, then the string is "ï»¿ABC". If you believe those bytes were encoded using UTF-8, then it is "ABC". If you don't know, then you must try to find out. The BOM could be a clue. The absence of invalid character when decoded as UTF-8 could be another... In the end, unless you can memorize/find the encoding somehow, an array of bytes is just an array of bytes. – paercebal Sep 11, 2011 at 18:57

24 @paercebal While "ï»¿" is valid latin-1, it is *very* unlikely that a text file begins with that combination. The same holds for the ucs2-le/be markers ÿþ and þÿ. Also you can *never* know. – user877329 Jun 21, 2013 at 16:48

17 @deceze It is probably linguistically invalid: First ï (which is ok), then some quotation mark without space in-between (not ok). ¿ indicates it is Spanish but ï is not used in Spanish. Conclusion: It is not latin-1 with a certainty well above the certainty without it. – user877329 Nov 5, 2013 at 7:20

29 @user Sure, it doesn't necessarily make sense. But if your system relies on *guessing*, that's where uncertainties come in. Some malicious user submits text starting with these 3 letters on purpose, and your system suddenly assumes it's looking at UTF-8 with a BOM, treats the text as UTF-8 where it should use Latin-1, and some Unicode injection takes place. Just a hypothetical example, but certainly possible. You can't judge a text encoding by its content, period. – deceze ♦ Nov 5, 2013 at 7:44

57 **"Encodings should be known, not divined."** The heart and soul of the problem. +1, good sir. In other words: either standardize your content and say, "We're always using this encoding. Period. Write it that way. Read it that way," or develop an extended format that allows for storing the encoding as metadata. (The latter probably needs some "bootstrap standard encoding," too. Like saying "The part that tells you the encoding is always ASCII.") – jpmc26 Jul 23, 2015 at 21:25

182

There are at least three problems with putting a BOM in UTF-8 encoded files.

1. Files that hold no text are no longer empty because they always contain the BOM.
2. Files that hold text within the ASCII subset of UTF-8 are no longer themselves ASCII because the BOM is not ASCII, which makes some existing tools break down, and it can be impossible for users to replace such legacy tools.

3. It is not possible to concatenate several files together because each file now has a BOM at the beginning.

And, as others have mentioned, it is neither sufficient nor necessary to have a BOM to detect that something is UTF-8:

- It is not sufficient because an arbitrary byte sequence can happen to start with the exact sequence that constitutes the BOM.
- It is not necessary because you can just read the bytes as if they were UTF-8; if that succeeds, it is, by definition, valid UTF-8.

Share Follow

edited Sep 9, 2022 at 16:16
Henke - Нава́льный П с м
5,797 ● 6 ● 41 ● 51

answered Nov 15, 2012 at 13:28

jpsecher
4,879 ● 2 ● 36 ● 43

12 Re point 1 "Files that hold no text are no longer empty because they always contain the BOM", this (1) conflates the OS filesystem level with the interpreted contents level, plus it (2) incorrectly assumes that using BOM one must put a BOM also in every otherwise empty file. The practical solution to (1) is to not do (2). Essentially the complaint reduces to "it's possible to impractically put a BOM in an otherwise empty file, thus preventing the most easy detection of logically empty file (by checking file size)". Still good software should be able to deal with it, since it has a purpose. – Cheers and hth. - Alf Jun 18, 2014 at 14:22

11 Re point 2, "Files that hold ASCII text is no longer themselves ASCII", this conflates ASCII with UTF-8. An UTF-8 file that holds ASCII text is not ASCII, it's UTF-8. Similarly, an UTF-16 file that holds ASCII text is not ASCII, it's UTF-16. And so on. ASCII is a 7-bit single byte code. UTF-8 is an 8-bit variable length extension of ASCII. If "tools break down" due to >127 values then they're just not fit for an 8-bit world. One simple practical solution is to use only ASCII files with tools that break down for non-ASCII byte values. A probably better solution is to ditch those ungood tools. – Cheers and hth. - Alf Jun 18, 2014 at 14:27

9 Re point 3, "It is not possible to concatenate several files together because each file now has a BOM at the beginning" is just wrong. I have no problem concatenating UTF-8 files with BOM, so it's clearly possible. I think maybe you meant the Unix-land `cat` won't give you a *clean* result, a result that has BOM only at the start. If you meant that, then that's because `cat` works at the byte level, not at the interpreted contents level, and in similar fashion `cat` can't deal with photographs, say. Still it doesn't do much harm. That's because the BOM encodes a zero-width non-breaking space. – Cheers and hth. - Alf Jun 18, 2014 at 14:34

34 @Cheersandhth.-Alf This answer is correct. You are merely pointing out Microsoft bugs. – tchrist Oct 1, 2014 at 22:34

15 @brighty: The situation isn't improved any by adding a bom though. – Deduplicator Sep 20, 2015 at 4:29

144

Here are examples of the BOM usage that actually cause real problems and yet many people don't know about it.

BOM breaks scripts

Shell scripts, Perl scripts, Python scripts, Ruby scripts, Node.js scripts or any other executable that needs to be run by an interpreter - all start with a shebang line which looks like one of those:

```
#!/bin/sh
#!/usr/bin/python
#!/usr/local/bin/perl
#!/usr/bin/env node
```

It tells the system which interpreter needs to be run when invoking such a script. If the script is encoded in UTF-8, one may be tempted to include a BOM at the beginning. But actually the "#!" characters are not just characters. They are in fact a magic number that happens to be composed out of two ASCII characters. If you put something (like a BOM) before those characters, then the file will look like it had a different magic number and that can lead to problems.

See Wikipedia, article: Shebang, section: Magic number:

> The shebang characters are represented by the same two bytes in extended ASCII encodings, including UTF-8, which is commonly used for scripts and other text files on current Unix-like systems. However, UTF-8 files may begin with the optional byte order mark (BOM); if the "exec" function specifically detects the bytes 0x23 and 0x21, then **the presence of the BOM (0xEF 0xBB 0xBF) before the shebang will prevent the script interpreter from being executed.** Some authorities recommend against using the byte order mark in POSIX (Unix-like) scripts,[14] for this reason and for wider interoperability and philosophical concerns. Additionally, a byte order mark is not necessary in UTF-8, as that encoding does not have endianness issues; it serves only to identify the encoding as UTF-8. [emphasis added]

BOM is illegal in JSON

See RFC 7159, Section 8.1:

> **Implementations MUST NOT add a byte order mark to the beginning of a JSON text.**

BOM is redundant in JSON

Not only it is **illegal** in JSON, it is also **not needed** to determine the character encoding because there are more reliable ways to unambiguously determine both the character encoding and endianness used in any JSON stream (see this answer for details).

BOM breaks JSON parsers

Not only it is **illegal** in JSON and **not needed**, it actually **breaks all software** that determine the encoding using the method presented in RFC 4627:

Determining the encoding and endianness of JSON, examining the first four bytes for the NUL byte:

```
00 00 00 xx - UTF-32BE
00 xx 00 xx - UTF-16BE
xx 00 00 00 - UTF-32LE
xx 00 xx 00 - UTF-16LE
xx xx xx xx - UTF-8
```

Now, if the file starts with BOM it will look like this:

```
00 00 FE FF - UTF-32BE
FE FF 00 xx - UTF-16BE
FF FE 00 00 - UTF-32LE
FF FE xx 00 - UTF-16LE
EF BB BF xx - UTF-8
```

Note that:

1. UTF-32BE doesn't start with three NULs, so it won't be recognized
2. UTF-32LE the first byte is not followed by three NULs, so it won't be recognized
3. UTF-16BE has only one NUL in the first four bytes, so it won't be recognized
4. UTF-16LE has only one NUL in the first four bytes, so it won't be recognized

Depending on the implementation, all of those may be interpreted incorrectly as UTF-8 and then misinterpreted or rejected as invalid UTF-8, or not recognized at all.

Additionally, if the implementation tests for valid JSON as I recommend, it will reject even the input that is indeed encoded as UTF-8, because it doesn't start with an ASCII character < 128 as it should according to the RFC.

Other data formats

BOM in JSON is not needed, is illegal and breaks software that works correctly according to the RFC. It should be a nobrainer to just not use it then and yet, there are always people who insist on breaking JSON by using BOMs, comments, different quoting rules or different data types. Of course anyone is free to use things like BOMs or anything else if you need it - just don't call it JSON then.

For other data formats than JSON, take a look at how it really looks like. If the only encodings are UTF-* and the first character must be an ASCII character lower than 128 then you already have all

the information needed to determine both the encoding and the endianness of your data. Adding BOMs even as an optional feature would only make it more complicated and error prone.

Other uses of BOM

As for the uses outside of JSON or scripts, I think there are already very good answers here. I wanted to add more detailed info specifically about scripting and serialization, because it is an example of BOM characters causing real problems.

Share Follow

edited Oct 7, 2021 at 7:34
Community Bot
1 • 1

answered Jun 26, 2016 at 11:34

rsp
112k • 31 • 209 • 183

7 rfc7159 which supersedes rfc4627 actually suggests supporting BOM may not be so evil. Basically not having a BOM is just an ambiguous kludge so that old Windows and Unix software that are not Unicode-aware can still process utf-8. – Eric Grange Apr 10, 2017 at 7:59

24 @EricGrange, you seem to be very strongly supporting BOM, but fail to realize that this would render the all-ubiquitous, universally useful, *optimal-minimum* "plain text" format a relic of the pre-UTF8 past! Adding any sort of (in-band) header to the *plain* text stream would, by definition, *impose a mandatory protocol* to the simplest text files, making it never again the "simplest"! And for what gain? To support all the *other*, ancient CP encodings that *also* didn't have signatures, so you might mistake them with UTF-8? (BTW, ASCII is UTF-8, too. So, a BOM to those, too? ;) Come on.) – Sz. Mar 14, 2018 at 22:20

6 This answer is the reason why I came up to this question! I creat my bash scripts in Windows and experience a lot of problems when publishing those scripts to Linux! Same thing with jason files. – Tono Nam Jul 2, 2019 at 14:43

5 I wish I could vote this answer up about fifty times. I also want to add that at this point, UTF-8 has won the standards war, and nearly all text being produced on the Internet is UTF-8. Some of the most popular programming languages (such as C# and Java) use UTF-16 internally, but when programmers using those languages write files to output streams, they almost always encode them as UTF-8. Therefore, it no longer makes sense to have a BOM to mark a UTF-8 file; UTF-8 should be the default you use when reading, and only try other encodings if UTF-8 decoding fails. – rmunn Aug 23, 2019 at 1:56

5 @EricGrange, your answer, if you really studied it, is a bit disingenuous. You didn't include a link for rfc7159. Had you done so, people could have read: "Implementations MUST NOT add a byte order mark to the beginning of a JSON text. In the interests of interoperability, implementations that parse JSON texts MAY ignore the presence of a byte order mark rather than treating it as an error." That doesn't "[suggest] supporting BOM may not be so evil", it suggests that wise coders not make their programs break due to the Microsoft-created UTF8-with-BOM. – bballdave025 Jan 17, 2020 at 23:10

52

> What's different between UTF-8 and UTF-8 without BOM?

Short answer: In UTF-8, a BOM is encoded as the bytes `EF BB BF` at the beginning of the file.

Long answer:

Originally, it was expected that Unicode would be encoded in UTF-16/UCS-2. The BOM was designed for this encoding form. When you have 2-byte code units, it's necessary to indicate which order those two bytes are in, and a common convention for doing this is to include the character U+FEFF as a "Byte Order Mark" at the beginning of the data. The character U+FFFE is permanently unassigned so that its presence can be used to detect the wrong byte order.

UTF-8 has the same byte order regardless of platform endianness, so a byte order mark isn't needed. However, it may occur (as the byte sequence `EF BB FF`) in data that was converted to UTF-8 from UTF-16, or as a "signature" to indicate that the data is UTF-8.

> Which is better?

Without. As Martin Cote answered, the Unicode standard does not recommend it. It causes problems with non-BOM-aware software.

A better way to detect whether a file is UTF-8 is to perform a validity check. UTF-8 has strict rules about what byte sequences are valid, so the probability of a false positive is negligible. If a byte sequence looks like UTF-8, it probably is.

Share Follow

edited May 6, 2015 at 19:27

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Jul 31, 2010 at 22:53

dan04
91.3k ● 23 ● 169 ● 204

9 this would also invalidate valid UTF-8 with a single erroneous byte in it, though :/ – endolith Jul 15, 2012 at 1:05

10 -1 re " It causes problems with non-BOM-aware software.", that's never been a problem for me, but on the contrary, that absence of BOM causes problems with BOM-aware software (in particular Visual C++) has been a problem. So this statement is very **platform-specific**, a narrow Unix-land point of view, but is misleadingly presented as if it applies in general. Which it does not. – Cheers and hth. - Alf Jun 18, 2014 at 14:46

6 No, UTF-8 has no BOM. This answer is incorrect. See the Unicode Standard. – tchrist Oct 1, 2014 at 22:35

2 You can even think you have a pure ASCII file when just looking at the bytes. But this could be a utf-16 file as well where you'd have to look at words and not at bytes. Modern sofware should be aware about BOMs. Still reading utf-8 can fail if detecting invalid sequences, codepoints that can use a smaller sequence or codepoints that are surrogates. For utf-16 reading might fail too when there are orphaned surrogates. – brighty Feb 9, 2015 at 16:56

3 @Alf, I disagree with your interpretation of a non-BOM attitude as "**platform-specific**, a narrow Unix-land point of view." To me, the only way that the narrow-mindedness could lie with "Unix land" were if MS and Visual C++ came before *NIX, which they didn't. The fact that MS (I assume knowingly) started using a BOM in UTF-8 rather than UTF-16 suggests to me that they promoted breaking `sh`, `perl`, `g++`, and many other free and powerful tools. Want things to work? Just **buy** the MS versions. MS created the platform-specific problem, just like the disaster of their \x80-\x95 range. – bballdave025 Jan 17, 2020 at 23:17

37

UTF-8 with BOM is better identified. I have reached this conclusion the hard way. I am working on a project where one of the results is a CSV file, including Unicode characters.

If the CSV file is saved without a BOM, Excel thinks it's ANSI and shows gibberish. Once you add "EF BB BF" at the front (for example, by re-saving it using Notepad with UTF-8; or Notepad++ with UTF-8 with BOM), Excel opens it fine.

Prepending the BOM character to Unicode text files is recommended by RFC 3629: "UTF-8, a transformation format of ISO 10646", November 2003 at https://www.rfc-editor.org/rfc/rfc3629 (this last info found at: http://www.herongyang.com/Unicode/Notepad-Byte-Order-Mark-BOM-FEFF-EFBBBF.html)

Share Follow

edited Oct 7, 2021 at 5:46
Community Bot
1 • 1

answered Jun 28, 2012 at 17:34
Helen Craigman
1,453 • 3 • 16 • 26

7 Thanks for this excellent tip in case one is creating UTF-8 files for use by Excel. In other circumstances though, I would still follow the other answers and skip the BOM. – barfuin May 7, 2013 at 19:20

5 It's also useful if you create files that contain only ASCII and later may have non-ascii added to it. I have just ran into such an issue: software that expects utf8, creates file with some data for user editing. If the initial file contains only ASCII, is opened in some editors and then saved, it ends up in latin-1 and everything breaks. If I add the BOM, it will get detected as UTF8 by the editor and everything works. – Roberto Alsina Sep 9, 2013 at 22:03

1 I have found multiple programming related tools which require the BOM to properly recognise UTF-8 files correctly. Visual Studio, SSMS, SoureTree.... – kjbartel Jan 27, 2015 at 13:24

10 **Where do you read a *recommendation* for using a BOM into that RFC?** At most, there's a strong recommendation to not forbid it under certain circumstances where doing so is difficult. – Deduplicator Aug 11, 2015 at 18:37

15 *Excel thinks it's ANSI and shows gibberish* then the problem is in Excel. – user8017719 Nov 26, 2016 at 8:10

20

> **Question:** What's different between UTF-8 and UTF-8 without a BOM? Which is better?

Here are some excerpts from the Wikipedia article on the byte order mark (BOM) that I believe offer a solid answer to this question.

On the meaning of the BOM and UTF-8:

> The Unicode Standard permits the **BOM** in **UTF-8**, but does not require or recommend its use. Byte order has no meaning in UTF-8, so its only use in UTF-8 is to signal at the start that the text stream is encoded in UTF-8.

Argument for *NOT* using a BOM:

> The primary motivation for not using a BOM is backwards-compatibility with software that is not Unicode-aware... Another motivation for not using a BOM is to encourage UTF-8 as the "default" encoding.

Argument *FOR* using a BOM:

> The argument for using a BOM is that without it, heuristic analysis is required to determine what character encoding a file is using. Historically such analysis, to distinguish various 8-bit encodings, is complicated, error-prone, and sometimes slow. A number of libraries are available to ease the task, such as Mozilla Universal Charset Detector and International Components for Unicode.
>
> Programmers mistakenly assume that detection of UTF-8 is equally difficult (it is not because of the vast majority of byte sequences are invalid UTF-8, while the encodings these libraries are trying to distinguish allow all possible byte sequences). Therefore not all Unicode-aware programs perform such an analysis and instead rely on the BOM.
>
> In particular, **Microsoft** compilers and interpreters, and many pieces of software on Microsoft Windows such as Notepad will not correctly read UTF-8 text unless it has only ASCII characters or it starts with the BOM, and will add a BOM to the start when saving text as UTF-8. Google Docs will add a BOM when a Microsoft Word document is downloaded as a plain text file.

On which is better, *WITH* or *WITHOUT* the BOM:

> The IETF recommends that if a protocol either (a) always uses UTF-8, or (b) has some other way to indicate what encoding is being used, then it “SHOULD forbid use of U+FEFF as a signature.”

My Conclusion:

Use the BOM ***only*** if compatibility with a software application is absolutely essential.

Also note that while the referenced Wikipedia article indicates that many Microsoft applications rely on the BOM to correctly detect UTF-8, this is not the case for *all* Microsoft applications. For example, as pointed out by @barlop, when using the Windows Command Prompt with UTF-8[†], commands such `type` and `more` do not expect the BOM to be present. If the BOM *is* present, it can be problematic as it is for other applications.

[†] The `chcp` command offers support for UTF-8 (*without* the BOM) via code page 65001.

Share Follow

edited Mar 4, 2018 at 1:16

answered Oct 2, 2014 at 20:24

DavidRR
19.5k ● 27 ● 111 ● 196

5 I'd better to strict to **WITHOUT the BOM**. I found that `.htaccess` and `gzip compression` in combination with UTF-8 BOM gives an encoding error Change to Encoding in UTF-8 without BOM follow to a suggestion as explained here solve the problems – eQ19 Apr 16, 2015 at 15:09

1 *'Another motivation for not using a BOM is to encourage UTF-8 as the "default" encoding.'* -- Which is so strong & valid an argument, that you could have actually stopped the answer there!... ;-o Unless you got a better idea for universal text representation, that is. ;) (I don't know how old you are, how many years you had to suffer in the pre-UTF8 era (when linguists desperately considered even changing their alphabets), but I can tell you that every second we get closer to ridding the mess of all the ancient single-byte-with-no-metadata encodings, instead of having "the one" is pure joy.) – Sz. Mar 14, 2018 at 22:41

See also this comment about how adding a BOM (or anything!) to the simplest of the text file formats, "plain text", would mean preventing exactly *the best universal text encoding format* from being "plain", and "simple" (i.e. "overheadless")!... – Sz. Mar 14, 2018 at 22:58

BOM is mostly problematic on Linux because many utilities do not really support Unicode to begin with (they will happily truncate in the middle of codepoints for instance). For most other modern software environment, use BOM whenever the encoding is not unambiguous (through specs or metadata).
– Eric Grange Aug 23, 2019 at 7:58

19

This question already has a million-and-one answers and many of them are quite good, but I wanted to try and clarify when a BOM should or should not be used.

As mentioned, any use of the UTF BOM (Byte Order Mark) in determining whether a string is UTF-8 or not is educated guesswork. If there is proper metadata available (like `charset="utf-8"`), then you already know what you're supposed to be using, but otherwise you'll need to test and make some assumptions. This involves checking whether the file a string comes from begins with the hexadecimal byte code, EF BB BF.

If a byte code corresponding to the UTF-8 BOM is found, the probability is high enough to assume it's UTF-8 and you can go from there. When forced to make this guess, however, additional error checking while reading would still be a good idea in case something comes up garbled. You should only assume a BOM is not UTF-8 (i.e. latin-1 or ANSI) if the input *definitely shouldn't be* UTF-8 based on its source. If there is no BOM, however, you can simply determine whether it's supposed to be UTF-8 by validating against the encoding.

Why is a BOM not recommended?

1. Non-Unicode-aware or poorly compliant software may assume it's latin-1 or ANSI and won't strip the BOM from the string, which can obviously cause issues.
2. It's not really needed (just check if the contents are compliant and always use UTF-8 as the fallback when no compliant encoding can be found)

When *should* you encode with a BOM?

If you're unable to record the metadata in any other way (through a charset tag or file system meta), and the programs being used like BOMs, you should encode with a BOM. This is especially true on Windows where anything without a BOM is generally assumed to be using a legacy code page. The BOM tells programs like Office that, yes, the text in this file is Unicode; here's the encoding used.

When it comes down to it, the only files I ever really have problems with are CSV. Depending on the program, it either must, or must not have a BOM. For example, if you're using Excel 2007+ on Windows, it must be encoded with a BOM if you want to open it smoothly and not have to resort to importing the data.

Share Follow

edited Apr 16, 2020 at 23:37

Peter Mortensen
31.6k ●22 ●110 ●133

answered Jan 25, 2016 at 16:03

jpc-ae
191 ●1 ●5

11 The last section of your answer is 100% correct: the *only* reason to use a BOM is when you have to interoperate with buggy software that doesn't use UTF-8 as its default to parse unknown files. – rmunn Aug 23, 2019 at 2:01

18

BOM tends to boom (no pun intended (sic)) somewhere, someplace. And when it booms (for example, doesn't get recognized by browsers, editors, etc.), it shows up as the weird characters `ï»¿` at the start of the document (for example, HTML file, JSON response, RSS, etc.) and causes the kind of embarrassments like the recent encoding issue experienced during the talk of Obama on Twitter.

It's very annoying when it shows up at places hard to debug or when testing is neglected. So it's best to avoid it unless you must use it.

Share Follow

edited May 6, 2015 at 19:28

Peter Mortensen
31.6k ●22 ●110 ●133

answered Jul 11, 2011 at 7:56

Halil Özgür
15.9k ●6 ●50 ●58

Yes, just spent hours identifying a problem caused by a file being encoded as UTF-8 instead of UTF-8 without BOM. (The issue only showed up in IE7 so that led me on a quite a goose chase. I used Django's "include".) – user984003 Jan 31, 2013 at 20:45

Future readers: Note that the tweet issue I've mentioned above was not strictly related to BOM, but if it was, then the tweet would be garbled in a similar way, but at the start of the tweet. – Halil Özgür Feb 1, 2013 at 7:26

15 @user984003 No, the problem is that Microsoft has mislead you. What it calls UTF-8 is not UTF-8. What it calls UTF-8 without BOM is what UTF-8 really is. – tchrist Oct 2, 2014 at 0:11

what does the "sic" add to your "no pun intended" – JoelFan Oct 23, 2017 at 21:15

2 @JoelFan I can't recall anymore but I guess the pun might have been intended despite the author's claim :) – Halil Özgür Oct 23, 2017 at 21:34

9

UTF-8 without BOM has no BOM, which doesn't make it any better than UTF-8 with BOM, except when the consumer of the file needs to know (or would benefit from knowing) whether the file is UTF-8-encoded or not.

The BOM is usually useful to determine the endianness of the encoding, which is not required for most use cases.

Also, the BOM can be unnecessary noise/pain for those consumers that don't know or care about it, and can result in user confusion.

Share Follow

edited Feb 8, 2010 at 18:42

answered Feb 8, 2010 at 18:30

Romain

12.8k ● 3 ● 43 ● 54

3 "which has no use for UTF-8 as it is 8-bits per glyph anyway." Er... no, only ASCII-7 glyphs are 8-bits in UTF-8. Anything beyond that is going to be 16, 24, or 32 bits. – Powerlord Feb 8, 2010 at 18:38

5 "The BOM is usually useful to determine the endianness of the encoding, which is not required for most use cases."... endianness simply does not apply to UTF-8, regardless of use case – JoelFan Oct 23, 2017 at 21:30

a consumer that needs to know is broken by design,. – Jasen Aug 9, 2020 at 8:38

9

It should be noted that for some files you **must not** have the BOM even on Windows. Examples are `SQL*plus` or `VBScript` files. In case such files contains a BOM you get an error when you try to execute them.

Share Follow

edited Aug 11, 2015 at 18:43

Deduplicator

45.7k ● 7 ● 72 ● 123

answered Jan 31, 2015 at 21:09

Wernfried Domscheit

59.7k ● 10 ● 90 ● 127

8

Quoted at the bottom of the Wikipedia page on BOM: http://en.wikipedia.org/wiki/Byte-order_mark#cite_note-2

> "Use of a BOM is neither required nor recommended for UTF-8, but may be encountered in contexts where UTF-8 data is converted from other encoding forms that use a BOM or where the BOM is used as a UTF-8 signature"

Share Follow

answered Feb 8, 2010 at 18:35

pib
3,323 ● 20 ● 15

2 Do you have any example where software makes a decision of whether to use UTF-8 with/without BOM, based on whether the previous encoding it is encoding from, had a BOM or not?! That seems like an absurd claim – barlop Mar 3, 2018 at 15:31

7

UTF-8 with BOM only helps if the file actually contains some non-ASCII characters. If it is included and there aren't any, then it will possibly break older applications that would have otherwise interpreted the file as plain ASCII. These applications will definitely fail when they come across a non ASCII character, so in my opinion the BOM should only be added when the file can, and should, no longer be interpreted as plain ASCII.

I want to make it clear that I prefer to not have the BOM at all. Add it in if some old rubbish breaks without it, and replacing that legacy application is not feasible.

Don't make anything expect a BOM for UTF-8.

Share Follow

edited Apr 16, 2020 at 23:15

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Jul 3, 2014 at 2:43

James Wakefield
526 ● 3 ● 11

1 it's not certain that non UTF8-aware applications will fail if they encounter UTF8, the whole point of UTF8 is that many things will just work `wc(1)` will give a correct line and octet count, and a correct word count if no unicode-only spacing characters are used. – Jasen Aug 9, 2020 at 8:37

1 I agree with you @Jasen. Trying to workout if I just delete this old answer. My current opinion is that the answer is simply don't add a BOM. The end user can append one if they have to hack a file to make it work with old software. We shouldn't make software that perpetuates this incorrect behaviour. There is no reason why a file couldn't start with a zero-width-non-joiner that is meant to be interpreted as one. – James Wakefield Dec 16, 2021 at 4:31

6

I look at this from a different perspective. I think **UTF-8 with BOM is better** as it provides more information about the file. I use UTF-8 without BOM only if I face problems.

I am using multiple languages (even Cyrillic) on my pages for a long time and when the files are saved without BOM and I re-open them for editing with an editor (as cherouvim also noted), some characters are corrupted.

Note that Windows' classic Notepad automatically saves files with a BOM when you try to save a newly created file with UTF-8 encoding.

I personally save server side **scripting files (.asp, .ini, .aspx) with BOM** and **.html files without BOM**.

Share Follow

edited May 23, 2017 at 11:55 Community Bot 1 • 1

answered May 11, 2012 at 8:34 user1358065 103 • 1 • 4

4 Thanks for the excellent tip about windows classic Notepad. I already spent some time finding out the exact same thing. My consequence was to always use Notepad++ instead of windows classic Notepad. :-) – barfuin May 7, 2013 at 19:22

You better use madedit. It's the only Editor that - in hex mode - shows one character if you select a utf-8 byte sequence instead of a 1:1 Basis between byte and character. A hex-Editor that is aware about a UTF-8 file should bevave like madedit does! – brighty Feb 9, 2015 at 16:49

@brighty I don't think you need one to one for the sake of the BOM. it doesn't matter, it doesn't take much to recognise a utf-8 BOM is efbbbf or fffe (of fffe if read wrong). One can simply delete those bytes. It's not bad though to have a mapping for the rest of the file though, but to also be able to delete byte by byte too – barlop Mar 3, 2018 at 15:34

@barlop Why would you want to delete a utf-8 BOM if the file's content is utf-8 encoded? The BOM is recognized by modern Text Viewers, Text Controls as well as Text Editors. A one to one view of a utf-8 sequence makes no sense, since n bytes result in one character. Of course a text-editor or hex-editor should allow to delete any byte, but this can lead to invalid utf-8 sequences. – brighty Mar 4, 2018 at 16:41

@brighty utf-8 with bom is an encoding, and utf-8 without bom is an encoding. The cmd prompt uses utf8 without bom.. so if you have a utf8 file, you run the command `chcp 65001` for utf8 support, it's utf8 without bom. If you do `type myfile` it will only display properly if there is no bom. If you do `echo aaa>a.a` or `echo אאא>a.a` to output the chars to file a.a, and you have chcp 65001, it will output with no BOM. – barlop Mar 5, 2018 at 4:55

6

When you want to display information encoded in UTF-8 you may not face problems. Declare for example an HTML document as UTF-8 and you will have everything displayed in your browser that is contained in the body of the document.

But this is not the case when we have text, CSV and XML files, either on Windows or Linux.

For example, a text file in Windows or Linux, one of the easiest things imaginable, it is not (usually) UTF-8.

Save it as XML and declare it as UTF-8:

```
<?xml version="1.0" encoding="UTF-8"?>
```

It will not display (it will not be be read) correctly, even if it's declared as UTF-8.

I had a string of data containing French letters, that needed to be saved as XML for syndication. Without creating a UTF-8 file from the very beginning (changing options in IDE and "Create New File") or adding the BOM at the beginning of the file

```
$file="\xEF\xBB\xBF".$string;
```

I was not able to save the French letters in an XML file.

Share Follow

edited May 6, 2015 at 19:33

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Sep 10, 2012 at 16:50

Florin Sima
1,515 ● 17 ● 13

4 I know this is an old answer, but I just want to mention that it's wrong. Text files on Linux (can't speak for other Unixes) usually /are/ UTF-8. – Functino Nov 14, 2015 at 23:41

Is this answer for or against the UTF-8 BOM? I think it is *for*, but I can't be sure. – bballdave025 Nov 7, 2023 at 21:49

6

One practical difference is that if you write a shell script for Mac OS X and save it as plain UTF-8, you will get the response:

```
#!/bin/bash: No such file or directory
```

in response to the shebang line specifying which shell you wish to use:

```
#!/bin/bash
```

If you save as UTF-8, no BOM (say in BBEdit) all will be well.

Share Follow

edited May 6, 2015 at 19:46

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Jan 24, 2014 at 20:38

David
1,050 ● 1 ● 13 ● 23

12 That's because Microsoft has swapped the meaning of what the standard says. UTF-8 has no BOM: they have created ***Microsoft UTF-8*** which inserts a spurious BOM in front of the data stream and then told you that no, this is actually UTF-8. It is not. It is just extending and corrupting. – tchrist Oct 2, 2014 at 0:14

5

As mentioned above, UTF-8 with BOM may cause problems with non-BOM-aware (or compatible) software. I once edited HTML files encoded as UTF-8 + BOM with the Mozilla-based KompoZer, as a client required that WYSIWYG program.

Invariably the layout would get destroyed when saving. It took my some time to fiddle my way around this. These files then worked well in Firefox, but showed a CSS quirk in Internet Explorer destroying the layout, again. After fiddling with the linked CSS files for hours to no avail I discovered that Internet Explorer didn't like the BOMfed HTML file. Never again.

Also, I just found this in Wikipedia:

> The shebang characters are represented by the same two bytes in extended ASCII encodings, including UTF-8, which is commonly used for scripts and other text files on current Unix-like systems. However, UTF-8 files may begin with the optional byte order mark (BOM); if the "exec" function specifically detects the bytes 0x23 0x21, then the presence of the BOM (0xEF 0xBB 0xBF) before the shebang will prevent the script interpreter from being executed. Some authorities recommend against using the byte order mark in POSIX (Unix-like) scripts,[15] for this reason and for wider interoperability and philosophical concerns

Share Follow

edited May 6, 2015 at 19:44

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Jun 22, 2013 at 4:56

Marek Möhling
152 ● 1 ● 8

5

The Unicode Byte Order Mark (BOM) FAQ provides a concise answer:

> Q: How I should deal with BOMs?
>
> A: Here are some guidelines to follow:
>
> 1. A particular protocol (e.g. Microsoft conventions for .txt files) may require use of the BOM on certain Unicode data streams, such as files. When you need to conform to such a protocol, use a BOM.
> 2. Some protocols allow optional BOMs in the case of untagged text. In those cases,
>   - Where a text data stream is known to be plain text, but of unknown encoding, BOM can be used as a signature. If there is no BOM, the encoding could be anything.
>   - Where a text data stream is known to be plain Unicode text (but not which endian), then BOM can be used as a signature. If there is no BOM, the text should be interpreted as big-endian.
> 3. Some byte oriented protocols expect ASCII characters at the beginning of a file. If UTF-8 is used with these protocols, use of the BOM as encoding form signature should be avoided.
> 4. Where the precise type of the data stream is known (e.g. Unicode big-endian or Unicode little-endian), the BOM should not be used. In particular, whenever a data stream is declared to be UTF-16BE, UTF-16LE, UTF-32BE or UTF-32LE a BOM must not be used.

Share Follow

answered Mar 8, 2018 at 13:58

Wernfried Domscheit
59.7k ● 10 ● 90 ● 127

4

From http://en.wikipedia.org/wiki/Byte-order_mark:

> The byte order mark (BOM) is a Unicode character used to signal the endianness (byte order) of a text file or stream. Its code point is U+FEFF. BOM use is optional, and, if used, should appear at the start of the text stream. Beyond its specific use as a byte-order indicator, the BOM character may also indicate which of the several Unicode representations the text is encoded in.

Always using a BOM in your file will ensure that it always opens correctly in an editor which supports UTF-8 and BOM.

My real problem with the absence of BOM is the following. Suppose we've got a file which contains:

```
abc
```

Without BOM this opens as ANSI in most editors. So another user of this file opens it and appends some native characters, for example:

```
abg-αβγ
```

Oops... Now the file is still in ANSI and guess what, "αβγ" does not occupy 6 bytes, but 3. This is not UTF-8 and this causes other problems later on in the development chain.

Share Follow

edited May 6, 2015 at 19:23

Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Feb 8, 2010 at 18:31

cherouvim
31.9k ● 15 ● 106 ● 156

10 An ensure that spurious bytes appear in the beginning of non BOM-aware software. Yay. – Romain Feb 8, 2010 at 18:33

1 @Romain Muller: e.g. PHP 5 will throw "impossible" errors when you try to send headers after the BOM. – Piskvor left the building Feb 8, 2010 at 18:47

5 αβγ is not ascii, but can appear in 8bit-ascii-bassed encodings. The use of a BOM disables a benafit of utf-8, its compatability with ascii (ability to work with lagacy applications where pure ascii is used). – ctrl-alt-delor Jan 7, 2011 at 13:03

1 This is the wrong answer. A string with a BOM in front of it is something else altogether. It is not supposed to be there and just screws everything up. – tchrist Oct 2, 2014 at 0:13

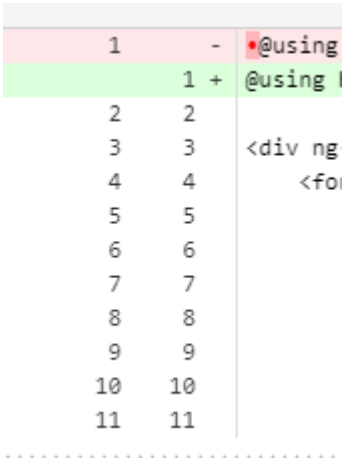
Without BOM this opens as ANSI in most editors. I agree absolutely. If this happens you're lucky if you deal with the correct Codepage but indeed it's just a guess, because the Codepage is not part of the file. A

BOM is. – brighty Feb 9, 2015 at 16:59

1

Here is my experience with Visual Studio, Sourcetree and Bitbucket pull requests, which has been giving me some problems:

So it turns out BOM with a signature will include a red dot character on each file when reviewing a pull request (it can be quite annoying).



If you hover on it, it will show a character like "ufeff", but it turns out Sourcetree does not show these types of bytemarks, so it will most likely end up in your pull requests, which should be ok because that's how Visual Studio 2017 encodes new files now, so maybe Bitbucket should ignore this or make it show in another way, more info here:

Red dot marker BitBucket diff view

Share Follow

edited Apr 16, 2020 at 23:47
Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Jul 31, 2019 at 9:30
Leo
1,027 ● 9 ● 26

1

I save a autohotkey file with utf-8, the chinese characters become strang.

With utf-8 BOM, works fine.

> AutoHotkey will not automatically recognize a UTF-8 file unless it begins with a byte order mark.

https://www.autohotkey.com/docs/FAQ.htm#nonascii

Share Follow

answered May 8, 2022 at 3:41
Good Pen
829 ● 8 ● 11

-5

UTF with a BOM is better if you use UTF-8 in HTML files and if you use Serbian Cyrillic, Serbian Latin, German, Hungarian or some exotic language on the same page.

That is my opinion (30 years of computing and IT industry).

Share Follow

edited Apr 16, 2020 at 23:11
Peter Mortensen
31.6k ● 22 ● 110 ● 133

answered Mar 15, 2013 at 10:01
user2173444
17

1 I find this to be true as well. If you use characters outside of the first 255 ASCII set and you omit the BOM, browsers interpret it as ISO-8859-1 and you get garbled characters. Given the answers above, this is apparently on the browser-vendors doing the wrong thing when they don't detect a BOM. But unless you work at Microsoft Edge/Mozilla/Webkit/Blink, you have no choice but work with the defects these apps have. – asontu Nov 28, 2017 at 8:42

1 UTF what? UTF-8? UTF-16? Something else? – Peter Mortensen Apr 16, 2020 at 23:12

1 If your server does nt indocate the correct mime type charset parameter you should use the `<meta http-equiv` tag in your HTML header. – Jasen Aug 9, 2020 at 8:49

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

unicode utf-8 character-encoding byte-order-mark

See similar questions with these tags.