

Clarify guidance for use of a BOM as a UTF-8 encoding signature

Abstract

The Unicode standard is clear that a BOM may be used as an encoding signature for UTF-8 encoded data, but its guidance regarding when a BOM is or is not recommended for such use is not consistently interpreted.

This paper seeks to clarify the guidance offered by the Unicode standard for use of a BOM as an encoding signature and proposes several possible resolutions ranging from removal of existing guidance to expanding guidance tailored to protocol designers, software developers, and text authors.

Introduction

This paper follows prior discussion on the unicode.org mailing list. The relevant email threads are archived and available at <https://corp.unicode.org/pipermail/unicode/2020-June/008713.html> and <https://corp.unicode.org/pipermail/unicode/2020-October/009070.html>.

Unicode 13, in the “Byte Order” subsection of section 2.6, “Encoding Schemes”, states:

... 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 23.8, Specials*, for more information.

This statement is sometimes interpreted as a recommendation against use of a BOM in UTF-8 text and sometimes interpreted as the lack of a recommendation either for or against such use. In either case, no rationale for a recommendation or lack thereof is offered within this section or in those it references.

Unicode 13, in the “Unicode Signature” subsection of section 2.13, “Special Characters”, states:

Unicode Signature. An initial BOM may also serve as an implicit marker to identify a file as containing Unicode text. For UTF-16, the sequence $FE_{16} FF_{16}$ (or its byte-reversed counterpart, $FF_{16} FE_{16}$) is exceedingly rare at the outset of text files that use other character encodings. The corresponding UTF-8 BOM sequence, $EF_{16} BB_{16} BF_{16}$, is also exceedingly rare. In either case, it is therefore unlikely to be confused with real text data. The same is true for both single-byte and multibyte encodings.

Data streams (or files) that begin with the U+FEFF byte order mark are likely to contain Unicode characters. **It is recommended that applications sending or receiving untyped data streams of coded characters use this signature. If other signaling methods are used, signatures should not be employed.**

Conformance to the Unicode Standard does not require the use of the BOM as such a signature. See *Section 23.8, Specials*, for more information on the byte order mark and its use as an encoding signature.

This section offers a clear recommendation favoring use of a BOM as an encoding signature in untyped data and a stronger recommendation against such use when encoding is indicated via other means. Neither interpretation of the highlighted text from section 2.6 is fully compatible with this section.

The referenced "Byte Order Mark" subsection of section 23.8 contains no guidance; it is factual and details some possible consequences of BOM use as an encoding signature, but does not apply a judgment. The following statements could be interpreted as an endorsement of such use in UTF-8 and other byte oriented encodings.

... Instead, its most common and most important usage is in the following two circumstances:

1. Unmarked Byte Order. ...
2. Unmarked Character Set. In some circumstances, the character set information for a stream of coded characters (such as a file) is not available. The only information available is that the stream contains text, but the precise character set is not known.

In these two cases, the character U+FEFF is used as a signature to indicate the byte order and the character set by using the byte serializations described in *Section 3.10, Unicode Encoding Schemes*. ...

In UTF-8, the BOM corresponds to the byte sequence $\langle EF_{16} BB_{16} BF_{16} \rangle$. Although there are never any questions of byte order with UTF-8 text, this sequence can serve as signature for UTF-8 encoded text where the character set is unmarked. ...

The characteristic sequences of bytes associated with an initial U+FEFF can serve as signatures in those cases, as shown in *Table 23-7*.

Table 23-7. U+FEFF Signature in Other Charsets

Charset	Signature
SCSU	0E FE FF
BOCU-1	FB EE 28
UTF-7	2B 2F 76 38 or 2B 2F 76 39 or 2B 2F 76 2B or 2B 2F 76 2F
UTF-EBCDIC	DD 73 66 73

The "Byte Order Mark (BOM)" section of the Unicode FAQ at https://www.unicode.org/faq/utf_bom.html#BOM can likewise be read as endorsing use of a BOM as an encoding signature. In this case, some guidance for use is provided.

Q: Where is a BOM useful?

A: A BOM is useful at the beginning of files that are typed as text, but for which it is not known whether they are in big or little endian format—it can also serve as a hint indicating that the file is in Unicode, as opposed to in a legacy encoding and furthermore, it act as a signature for the specific encoding form used.

Q: When a BOM is used, is it only in 16-bit Unicode text?

A: No, a BOM can be used as a signature no matter how the Unicode text is transformed: UTF-16, UTF-8, or UTF-32. The exact bytes comprising the BOM will be whatever the Unicode character U+FEFF is converted into by that transformation format. In that form, the BOM serves to indicate both that it is a Unicode file, and which of the formats it is in. ...

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. ...

The guidelines offered in the FAQ are targeted at text authors. How should a protocol designer or software developer interpret them? Should new protocols be designed to mandate use of a particular encoding and, if so, should the presence of a BOM be treated as an error? If a protocol requires UTF-8, but permits an optional BOM, should software targeting that protocol proactively suppress a BOM when copying text produced elsewhere? The guidelines are not clear on these questions.

The referenced sections do state some consequences for use of a BOM as an encoding signature in UTF-8, and those consequences could be used as rationale for avoidance as summarized below.

- Concatenating UTF-8 content containing a BOM requires that the BOM be removed in order to avoid unintended insertion of a U+FEFF character that then becomes part of the concatenated textual content.
- A BOM may interfere with normal processing of files that are required to begin with an ASCII sequence. POSIX shell scripts are an example where a BOM may interfere with recognition of a “#!” marker at the beginning of the file.
- Since a BOM is not required for endian determination, a BOM consumes space unnecessarily if the content is known to be UTF-8.

These consequences are too nuanced to provide clear guidance to text authors or developers of software or protocols that produce or consume UTF-8 content.

Proposed Resolutions

All proposed changes are relative to Unicode 13.

Option 1: Remove guidance regarding use of a BOM as a UTF-8 signature.

This would remove the existing guidance offered by the standard, presumably relegating such guidance to other standards or guidelines such as those in the Unicode FAQ.

Modify the “Byte Order” subsection of section 2.6, “Encoding Schemes” as follows:

The endian order entry for UTF-8 in *Table 2-4* is marked N/A because UTF-8 code units are 8 bits in size, and the usual machine issues of endian order for larger code units do not apply. The serialized order of the bytes must not depart from the order defined by the UTF-8 encoding form. Use of a BOM is ~~neither not~~ 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 23.8, Specials*, for more information. ~~The standard does not recommend for or against use of a BOM in UTF-8 data.~~

Modify the “Unicode Signature” subsection of section 2.13, “Special Characters” as follows:

Unicode Signature. An initial BOM may also serve as an implicit marker to identify a file as containing Unicode text. For UTF-16, the sequence FE₁₆ FF₁₆ (or its byte-reversed counterpart, FF₁₆ FE₁₆) is exceedingly rare at the outset of text files that use other character encodings. The corresponding UTF-8 BOM sequence, EF₁₆ BB₁₆ BF₁₆, is also exceedingly rare. In either case, it is therefore unlikely to be confused with real text data. The same is true for both single-byte and multibyte encodings.

Data streams (or files) that begin with the U+FEFF byte order mark are likely to contain Unicode characters. ~~It is recommended that applications sending or receiving untyped data streams of coded characters use this signature. If other signaling methods are used, signatures~~

~~should not be employed.~~

Conformance to the Unicode Standard does not require the use of the BOM as such a signature. See *Section 23.8, Specials*, for more information on the byte order mark and its use as an encoding signature.

Option 2: Expand the “Byte Order Mark (BOM)” subsection in section 23.8 to provide rationale and targeted guidance

The following is suggested wording to be added to section 23.8 to clarify the guidelines for use of a BOM as a UTF-8 encoding signature.

Guidelines for use of a BOM in UTF-8. The UTF-8 encoding scheme permits, but does not require, a BOM to be present. This raises the question of when a BOM should or should not be generated or expected when producing or consuming UTF-8 encoded text.

The utility of a BOM in UTF-8 is limited to scenarios in which a byte sequence contains text that may or may not be encoded as UTF-8. In such scenarios, a BOM may be useful to differentiate text encoded in one of a few possible character encodings. However, the presence of a BOM may also complicate text processing.

- Some text processing tools fail to handle BOMs correctly. This is especially true for programs that were historically encoding agnostic and for ad hoc programs written for one-time use purposes.
- A text processing tool must maintain additional state in order to recognize if an observed U+FEFF character is a BOM or whether it should be treated as a ZERO WIDTH NON-BREAKING SPACE (ZWNBS). Such state may or may not be intrinsic to the structure of the program.
- A text generating tool may be required to generate a BOM if the first character to be encoded is U+FEFF and that character is not intended to be used as a BOM. This is only required for compatibility with Unicode versions prior to 3.2; U+2060 WORD JOINER should be used in place of U+FEFF for such purposes with more recent Unicode versions.
- Concatenation of text containing a BOM requires care. When concatenating to an empty text, preservation of a BOM may be warranted, but otherwise, failure to elide the BOM will result in the insertion of a U+FEFF character that becomes part of the concatenated textual content.
- In situations where text is known to be encoded as UTF-8, a BOM consumes storage space unnecessarily. While this is unlikely to be a concern for a single document, it may be a significant concern in situations involving thousands or millions of small text sources.

Due to the above complications, use of a BOM as an encoding signature in UTF-8 text is discouraged. The following guidelines advise alternative approaches tailored for a few distinct audiences.

Except where otherwise noted, these approaches preclude the possibility of a text starting with a U+FEFF character that is not intended as a BOM under the expectation that such text is

exceedingly rare and most likely due to a failure to elide a BOM. Text authored for Unicode 3.2 or later should use U+2060 WORD JOINER instead.

Protocol designers:

- When designing a protocol that does **not** require compatibility with an existing corpus of differently encoded text, mandate use of UTF-8 without a BOM. Diagnose the presence of a BOM in consumed text as an error and produce UTF-8 text without a BOM. This approach is intended to explicitly discourage use of a BOM in UTF-8 text in new protocols so as to reduce requirement complexity for text processors.
- Otherwise, when designing a protocol that requires compatibility with an existing corpus of UTF-8 text that may contain BOMs, mandate use of UTF-8 with or without a BOM. Recognize and discard a BOM in consumed text and produce UTF-8 text without a BOM.
- Otherwise, when designing a protocol that requires compatibility with an existing corpus of differently encoded text, specify UTF-8 as the default encoding with use of other encodings negotiated using information other than a BOM. Recognize and discard a BOM in consumed text and produce UTF-8 text without a BOM.
- Otherwise, when designing a protocol that requires compatibility with an existing corpus of differently encoded ASCII based text, where the encoding of each text is not otherwise specified, and where UTF-8 cannot be assumed by default, mandate use of UTF-8 with a BOM. Recognize and retain a BOM in consumed text, produce UTF-8 text with a BOM unless the text contains no non-ASCII characters (the exception is intended to avoid the addition of a BOM to ASCII text thus rendering such text as non-ASCII). This approach should be reserved for scenarios in which UTF-8 cannot be adopted as a default due to backward compatibility concerns.
- Otherwise, the protocol must require compatibility with an existing corpus of differently encoded text, some of which may be non-ASCII based, and where UTF-8 cannot be specified as the default encoding. Mandate use of a BOM in all UTF-8 text. Recognize and retain a BOM in consumed text, produce UTF-8 text with a BOM, even in text that would otherwise be ASCII. This approach should be reserved as a last resort option in rare scenarios where non-ASCII based text is present.

Software developers:

- If consuming UTF-8, recognize and discard a BOM unless a protocol mandates the absence of a BOM in which case:
 - If compatibility with Unicode versions prior to 3.2 is a goal, treat the leading U+FEFF character as a ZERO WIDTH NON-BREAKING SPACE (ZWNBSpace).
 - Otherwise, diagnose the U+FEFF character as an error (a BOM where a BOM is not permitted).
- If producing UTF-8, include a BOM:
 - If explicitly directed to do so.
 - Otherwise, if a BOM is known to be required by a protocol.
 - Otherwise, if compatibility with Unicode versions prior to 3.2 is a goal and the text is intended to start with a U+FEFF character that is not intended as a BOM.

Text authors:

- Do not use U+FEFF as a ZWNBSP character; use U+2060 WORD JOINER instead.
- Include a BOM if one is known to be required by a targeted protocol.
- Otherwise, include a BOM when authoring a UTF-8 text file that contains non-ASCII characters, is not targeting a specific protocol, and may be opened by applications that will not assume UTF-8 by default (this is useful on systems like Microsoft Windows where some applications assume text files to be encoded with the Active Code Page).
- Otherwise, do not include a BOM.

Acknowledgements

Thank you to Peter Brett, Nial Douglas, Asmus Freytag, Jens Maurer, Alisdair Meredith, Murray Sargent, and Shawn Steele for their review and feedback on this paper.