Tom Honermann <tom@honermann.net> October 11th, 2020

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

This paper follows prior discussion on the unicode.org mailing list. The relevant email thread is archived and available at https://corp.unicode.org/pipermail/unicode/2020-June/008713.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.

That statement is unconditional regarding the recommendation against use of a BOM for UTF-8, but neither rationale nor guidance for the recommendation is provided.

The referenced "Byte Order Mark" subsection in section 23.8 contains no similar 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 <EF_{16} BB_{16} BF_{16}>. 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 given the recommendation in section 2.6 against use of a BOM in UTF-8? 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 questions like these.

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.

Possible Resolutions

Strike "nor recommended" from the quoted text of section 2.6

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.

> 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.

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 (ZWNBSP). 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:

- If possible, mandate use of UTF-8 without a BOM; diagnose the presence of a BOM in consumed text as an error, and produce text without a BOM.
- Otherwise, if possible, mandate use of UTF-8 with or without a BOM; accept and discard a BOM in consumed text, and produce text without a BOM.
- Otherwise, if possible, use UTF-8 as the default encoding with use of other encodings negotiated using information other than a BOM; accept and discard a BOM in consumed text, and produce text without a BOM.
- Otherwise, require the presence of a BOM to differentiate UTF-8 encoded text in both consumed and produced text unless the absence of a BOM would result in the text being interpreted as an ASCII-based encoding and the UTF-8 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.

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 (ZWNBSP).
  - 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 on a Microsoft Windows system (this is useful on Microsoft Windows systems where the default encoding used by most applications is not UTF-8).
- Otherwise, do not include a BOM.