DISPATCH Working Group
Internet-Draft
Intended status: Standards Track
Expires: 2 September 2026

C. Bormann
Universitt Bremen TZI

1 March 2026

Modern Network Unicode
draft-bormann-dispatch-modern-network-unicode-08

Abstract

BCP18 (RFC 2277) has been the basis for the handling of character-shaped data in IETF specifications for more than a quarter of a century now. It singles out UTF-8 (STD63, RFC 3629) as the “charset” that MUST be supported, and pulls in the Unicode standard with that.

Based on this, RFC 5198 both defines common conventions for the use of Unicode in network protocols and caters for the specific requirements of the legacy protocol Telnet. In applications that do not need Telnet compatibility, some of the decisions of RFC 5198 can be cumbersome.

The present specification defines “Modern Network Unicode” (MNU), which is a form of RFC 5198 Network Unicode that can be used in specifications that require the exchange of plain text over networks and where just mandating UTF-8 may not be sufficient, but there is also no desire to import all of the baggage of RFC 5198.

As characters are used in different environments, MNU is defined in a one-dimensional (1D) variant that is useful for identifiers and labels, but does not use a structure of text lines. A 2D variant is defined for text that is a sequence of text lines, such as plain text documents or markdown format. Additional variances of these two base formats can be used to tailor MNU to specific areas of application.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

(Insert embellished copy of abstract here, [BCP18][STD63][RFC5198].)

Complex specifications that use Unicode often come with detailed information on their Unicode usage; this level of detail generally is necessary to support some legacy applications. New, simple protocol specifications generally do not have such a legacy or need such details, but can instead simply use common practice, informed by decades of using Unicode. The present specification attempts to serve as a convenient reference for such protocol specifications, reducing their need for discussing Unicode to just pointing to the present specification and making a few simple choices.

There is no intention that henceforth all new protocols "must" use the present specification. It is offered as a standards-track specification simply so it can be normatively referenced from other standards-track specifications.

1.1. Conventions

Characters in this specification are named with their Unicode scalar value notated in the usual form U+NNNN (where NNNN is a four-to-six-digit upper case hexadecimal number giving the Unicode scalar value), or with their ASCII names (such as CR, LF, HT, RS, NUL) [STD80].

> | See [BCP137] for a more detailed discussion of ways to refer in
> | ASCII to Unicode characters (as well as to code points and code
> | units in various transformation formats).

General unsigned integer values written in hexadecimal are notated in the form 0xNNNN (where NNNN is one or more hexadecimal digits).

The following definitions apply:

Byte: 8-bit unit of information interchange, synonym for octet.

Character: Unicode scalar value, unless otherwise specified. (With [BCP18], this usage is generally appropriate for IETF specifications; Appendix A has more about the more complex models that Unicode uses for character-like entities.)

Please see Appendix A for some more detailed term definitions that may be helpful to relate this specification with the Unicode Standard; please do read its first paragraph if the definitions in this section seem inadequate.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [BCP14] (RFC2119) (RFC8174) when, and only when, they appear in all capitals, as shown here.

2. 1D Modern Network Unicode

One-dimensional Modern Network Unicode (1D MNU) is the form of Modern Network Unicode that can be used for one-dimensional text, i.e., text without a structure of separate text lines. It requires conformance to UTF-8 [STD63], as well as to the following four mandates:

1. Control codes (here specifically U+0000 to U+001F and U+007F to U+009F) MUST NOT be used. (Note that this also excludes line endings, so a 1D MNU text string cannot extend beyond a single line. See Section 3 below if line structure is needed.)

2. The characters U+2028 and U+2029 MUST NOT be used. (In case future Unicode versions add to the Unicode character categories Zl or Zp, any characters in these categories MUST NOT be used.)

3. Modern Network Unicode strongly RECOMMENDS that, except in unusual circumstances (see Appendix C), all text is transmitted in normalization form NFC. This mandate is not intended to be realized by routinely normalizing all text, but by encouraging text sources to use NFC if applicable.

4. The code points U+FFFE and U+FFFF MUST NOT be used. Also, Byte Order Marks (leading U+FEFF characters) MUST NOT be used.

Note that several control codes have been in use historically in ASCII documents even within a single text line. For instance BS (U+0008) and CR (U+000D) have been used as a crude way to combine (by overtyping) characters; the present specification does not support this behavior anymore. HT (U+0009, “TAB”) has been used for a form of compressing stretches of blank space; by keeping its actual definition open it has also been used to enable users of text to specify variable interpretations of blank space (“indentation”). The present specification RECOMMENDS not using a variance for 1D MNU that would enable the use of BS, HT, or CR; legacy usage of HT may be more appropriate in certain forms of 2D text.

3. 2D Modern Network Unicode

Two-dimensional Modern Network Unicode (2D MNU) enhances 1D MNU by structuring the text into lines. 2D MNU does this by using the control code LF (U+000A) for a line terminator. The use of an unterminated last line is permitted, but not encouraged. (This is not a variance.)

Historically, the Internet NVT (See Section "THE NETWORK VIRTUAL TERMINAL" in RFC 0854 as part of the Telnet specification [STD8]) and thus, much later, Network Unicode [RFC5198] have used the combination CR+LF as a line terminator, reflecting its heritage based on teletype functionality. The use of U+000D mostly introduces additional ways to create interoperability problems. [XML] goes to great lengths to reduce the harm the presence of CR characters does to interchange of XML documents. Pages 10 and 11 of [RFC0764] discuss how CR can actually only be used in the combinations CR+NUL and CR+LF. 2D MNU simply elides the CR.

In other words, 2D MNU adds the use of line endings, represented by a single LF character (which is then the only control character allowed). Variances are available for (1) allowing or even (2) requiring the use of CR before LF.

4. 3D?

Three-dimensional forms of text have been used, e.g., by using U+000C FF ("form feed") to add a page structure to the long-time ASCII-based RFC format [RFC2223], or by using U+001E RS ("record separator") to separate several (2D) JSON texts in a JSON sequence [RFC7464].

As the need for this form of structure is now mostly being addressed by building data structures around text items, the present specification does not define a common 3D MNU. If needed, variances such as the use of FF or RS can be described in the documents specifying the 3D format.

5. Variances

In addition to the basic 1D and 2D versions of MNU, this specification describes a number of variances that can be used in the forms such as "2D Modern Network Unicode with VVV", or "2D Modern Network Unicode with VVV, WWW, and YYY" for multiple variances used. Specifications that cannot directly use the basic MNU forms may be able to use MNU with one or more of these variances added.

An example for the usage of variances: Up to RFC 8649, RFCs were formatted in “2D Modern Network Unicode with CR-tolerant lines and FF characters”, or exceptionally [RFC8265], “2D Modern Network Unicode with CR-tolerant lines, FF characters, and BOM”.

5.1. With CR-tolerant lines

The variance “with CR-tolerant lines” allows the sequence CR LF as well as a single LF character as a line ending, ignoring the CR (i.e., the presence or absence of a CR in front of an LF MUST be equivalent). This may enable existing texts to be used as MNU without processing at the sender side (substituting that by processing at the receiver side). Note that, with this variance, a CR character cannot be used anywhere else but immediately preceding an LF character.

5.2. With CR-LF

The variance “with CR-LF lines” requires the sequence CR LF as a line ending; a single LF character is not allowed. This is appropriate for certain existing environments that have already made that determination.

5.3. With Line or Paragraph Separators

For 2D applications, and even for 1D applications that need to address text in 2D forms, it may be useful to enable the use of U+2028 and U+2029; this should be explicitly called out.

5.4. With HT Characters

In some cases, the use of HT characters (“TABs”) cannot be completely excluded. The variance “with HT characters” allows their use, without attempting to define their meaning (e.g., equivalence with spaces, column definitions, etc.).

5.5. With /CCC/ Characters

Some applications of MNU may need to add specific control characters, such as RS [RFC7464] or FF characters [RFC2223]. This variance is spelled with the ASCII name of the control character for CCC, e.g., “with RS characters”.

5.6. With NFC

This variance turns the encouragement of using NFC normalization form into a requirement. Please see Appendix C for why this should be an exceptional case.

5.7. With NFKC

Some applications require a stronger form of normalization than NFC. The variance “with NFKC” swaps out NFC and uses NFKC instead. This is probably best used in conjunction with “with Unicode version NNN”.

5.8. With Unicode Version NNN

Some applications need to be sure that a certain Unicode version is used. The variance “with Unicode version NNN” (where NNN is a Unicode version number) defines the Unicode version in use as NNN. Also, it requires that only characters assigned in that Unicode version are being used.

See Section 4 of [RFC5198] for more discussion of Unicode versions.

5.9. With BOM

In some environments, UTF-8 files need to be distinguished from files in other formats without the benefit of out of band information such as media types. One convention that has been used in certain environments was to prepend a “Byte Order Mark” (U+FEFF) as a distinguishing mark (as UTF-8 is not influenced by different byte orders, this mark does not serve a purpose in UTF-8). Sometimes these BOMs are included for interchange to ensure local copies of the file include the distinguishing mark. This variance enables this usage.

6. Using ABNF with Unicode

Internet STD 68, [STD68], defines Augmented BNF for Syntax Specifications: ABNF. Since the late 1970s, ABNF has often been used to formally describe the pieces of text that are meant to be used in an Internet protocol. ABNF was developed at a time when character coding grew more and more complicated, and even in its current form, discusses encoding of characters only briefly (Section 2.4 of RFC 5234 [STD68]). This discussion offers no information about how this should be used today (it actually still refers to 16-bit Unicode!).

The best current practice of using ABNF for Unicode-based protocols is as follows: ABNF is used as a grammar for describing sequences of Unicode code points, valued from 0x0 to 0x10FFFF. The actual encoding (as UTF-8) is never seen on the ABNF level; see Section 9.4 of [RFC6020] for a recent example of this. Approaches such as representing the rules of UTF-8 encoding in ABNF (see Section 3.5 of [RFC5255] for an example) add complexity without benefit and are NOT RECOMMENDED.

ABNF features such as case-insensitivity in literal text strings essentially do not work for general Unicode; text string literals therefore (and by the definition in Section 2.3 of RFC 5234 [STD68]) are limited to ASCII characters. That is often not actually a problem in text-based protocol definitions. Still, characters beyond ASCII need to be allowed in many productions. ABNF does not have access to Unicode character categories and thus will be limited in its expressiveness here. The core rules defines in Appendix B of RFC 5234 [STD68] are limited to ASCII as well; new rules will therefore need to be defined in any protocol employing modern Unicode.

The present specification recommends defining ABNF rules as in these examples:

```
; 1D modern unicode character:
uchar = %x20-7E ; exclude C0
      / %xA0-2027 ; exclude DEL, C1
      / %x202A-D7FF ; exclude U+2028/2029
      / %xE000-FFFD ; exclude U+FFFE/FFFF
      / %x10000-10FFFD ; could exclude more non-characters

; 2D modern unicode with lines -- newline:
unl = %x0A
u2dchar = unl / uchar
; alternatively, CR-tolerant newline:
ucrnl = [%x0D] %x0A
u2dcrchar = ucrnl / uchar

; if really needed, HT-tolerant 1D unicode character:
utchar = %x09 / uchar

; for applications that mostly are concerned about specifying details
; about using ASCII characters, but want to include the non-ASCII
; characters allowed in modern network unicode and its variances:
NONASCII = %xA0-D7FF / %xE000-10FFFD
```

7. IANA considerations

This specification places no requirements on IANA.

8. Security considerations

The security considerations of [RFC5198] apply.

A variance “with NUL characters” would create specific security considerations as discussed in the security considerations of [RFC5198] and should therefore only be used in circumstances that absolutely do require it.

9. References

9.1. Normative References

[BCP14] Best Current Practice 14,
<https://www.rfc-editor.org/info/bcp14>.
At the time of writing, this BCP comprises the following:

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[BCP18] Best Current Practice 18,
<https://www.rfc-editor.org/info/bcp18>.
At the time of writing, this BCP comprises the following:

Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <https://www.rfc-editor.org/info/rfc2277>.

[STD63] Internet Standard 63,
<https://www.rfc-editor.org/info/std63>.
At the time of writing, this STD comprises the following:

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[STD68] Internet Standard 68,
<https://www.rfc-editor.org/info/std68>.
At the time of writing, this STD comprises the following:

Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[STD80] Internet Standard 80,
<https://www.rfc-editor.org/info/std80>.
At the time of writing, this STD comprises the following:

Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

9.2. Informative References

[BCP137] Best Current Practice 137, <https://www.rfc-editor.org/info/bcp137>. At the time of writing, this BCP comprises the following:

Klensin, J., "ASCII Escaping of Unicode Characters", BCP 137, RFC 5137, DOI 10.17487/RFC5137, February 2008, <https://www.rfc-editor.org/info/rfc5137>.

[PRIVATE-USE] The Unicode Consortium, "Private-Use Characters, Noncharacters & Sentinels FAQ", n.d., <https://www.unicode.org/faq/private_use.html>.

[RFC0764] Postel, J., "Telnet Protocol specification", RFC 764, DOI 10.17487/RFC0764, June 1980, <https://www.rfc-editor.org/rfc/rfc764>.

[RFC2223] Postel, J. and J. Reynolds, "Instructions to RFC Authors", RFC 2223, DOI 10.17487/RFC2223, October 1997, <https://www.rfc-editor.org/rfc/rfc2223>.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008, <https://www.rfc-editor.org/rfc/rfc5198>.

[RFC5255] Newman, C., Gulbrandsen, A., and A. Melnikov, "Internet Message Access Protocol Internationalization", RFC 5255, DOI 10.17487/RFC5255, June 2008, <https://www.rfc-editor.org/rfc/rfc5255>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/rfc/rfc6020>.

[RFC7464] Williams, N., "JavaScript Object Notation (JSON) Text Sequences", RFC 7464, DOI 10.17487/RFC7464, February 2015, <https://www.rfc-editor.org/rfc/rfc7464>.

[RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265, DOI 10.17487/RFC8265, October 2017, <https://www.rfc-editor.org/rfc/rfc8265>.

[STD8] Internet Standard 8, <https://www.rfc-editor.org/info/std8>. At the time of writing, this STD comprises the following:

Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, DOI 10.17487/RFC0854, May 1983, <https://www.rfc-editor.org/info/rfc854>.

Postel, J. and J. Reynolds, "Telnet Option Specifications", STD 8, RFC 855, DOI 10.17487/RFC0855, May 1983, <https://www.rfc-editor.org/info/rfc855>.

[UNICODE] The Unicode Consortium, "The Unicode Standard: Version 17.0 — Core Specification", 9 September 2025, <https://www.unicode.org/versions/Unicode17.0.0/UnicodeStandard-17.0.pdf>. For convenience, this bibliography entry points to what was the most recent version of Unicode at the time of writing. It is, however, intended to be a generic reference to the most recent version of Unicode, which always can be found via http://www.unicode.org/versions/latest/ (http://www.unicode.org/versions/latest/).

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126/>.

Appendix A. Terminology

In order for the main body of this specification to stay readable for its target audience, we banish what could be considered excessive detail into appendices. Expert readers who know a lot more about Coded Character Sets and Unicode than the target audience of the present document is likely interested in, are requested to hold back the urge to request re-introduction of excessive detail into the main body. For instance, this specification has a local definition of “Unicode character” that is useful for protocol designers, but is not actually defined by the Unicode consortium; the details are here.

Some definitions below reference definitions given in the Unicode standard and are simplified from those, which see if the full detail is needed.

Grapheme: The smallest functional unit of a writing system. For computer representation, graphemes are represented by _characters_, sometimes a single character, sometimes decomposed into multiple characters. In visible presentation, graphemes are made visible as _glyphs_; for computer use, glyphs are collected into _typefaces_ (fonts). The term grapheme is used to abstract from the specific glyph used — a specific glyph from a typeface may be considered one of several _allographs_ for a grapheme.

Character: Only defined as “abstract character” in [UNICODE]: A unit of information used for the organization, control, or representation of textual data (definition D7). Most characters are used to stand for graphemes or to build graphemes out of several characters. Characters are usually used in ordered sequences, which are referred to as “character strings”.

Character set, Coded Character Set: A mapping from code points to characters. Note that the shorter form term “character set” is easily misunderstood for a set of characters constituting a palette to choose from; the latter are called “character repertoire”.

Character repertoire: A set of characters, in the sense of what is available as characters to choose from.

Code point: The (usually numeric) value used in a coded character set to refer to a character. Note that code points may undergo some encoding before interchange, see Unicode transformation format below. Code points can also be allocated to refer to internal constructs of a Unicode transformation format instead of referring to characters.

Unicode uses unsigned numbers between 0 and 1114111 (0x10FFFF) as code points.

Unicode transformation format (UTF): Character (and character strings) usually are interchanged as a sequence of bytes. A Unicode Transformation Format or Unicode Encoding Scheme specifies a byte serialization for a Unicode encoding form. (See definition D94 in Section 3.10, Unicode Encoding Schemes.)

Code unit: A bit combination used for encoding text for processing

or interchange. The Unicode Standard uses 8-bit code units in the UTF-8 encoding form, 16-bit code units in the UTF-16 encoding form, and 32-bit code units in the UTF-32 encoding form. (See Definition D77 in Section 3.9, Unicode Encoding Forms.)

Unicode scalar value: Unicode code points except high-surrogate and low-surrogate code points (see Appendix B). In other words, the ranges of integers 0 to 0xD7FF and 0xE000 to 0x10FFFF inclusive. (See Definition D76 in Section 3.9, Unicode Encoding Forms.)

Unicode character: For the purposes of the present specification, we use “Unicode character” as a synonym for “Unicode scalar value”. Only when enough context is present, the term may be abbreviated as “character”, the above more general definition notwithstanding.

Note that this term includes Unicode scalar values that are not Assigned Characters; for the purposes of a specification using the present document, it is rarely useful to make this distinction, as Unicode evolves and could assign characters not assigned. Note also that the definition of this term includes what Unicode terms “noncharacters”, which may be surprising if this term is taken at face value: Unicode uses it for one out of a stable set of 66 Unicode scalar values that have been set aside with the promise they will never be assigned. For a readable introduction into the historical context of this concept and the term, please see [PRIVATE-USE].

Appendix B. History, Legacy

Some of the complexity of using Unicode is, unsurprisingly, rooted in its long history of development.

Unicode originally was introduced around 1988 as a 16-bit character standard. By the early 1990s, a specification had been published, and a number of environments eagerly picked up Unicode. Unicode characters were represented as 16-bit values (UCS-2), enabling a simple character model using these uniformly 16-bit values as the characters. Programming language environments such as those of Java, JavaScript and C# (.NET) are now intimately entangled with this character model.

In the mid-1990s, it became clear that 16 bits would not suffice. Unicode underwent an extension to 31 bits, and then back to ~ 21 bits (0 to 0x10FFFF). For the 16-bit environments, this was realized by switching to UTF-16, a “Unicode transformation format” (UTF-16). This reassigned some of the 16-bit code points not for characters, but for 2 sets of 1024 “surrogates” that would be used in pairs to represent characters that do not fit into 16 bits. Each of these

surrogates supplies 10 bits; together they add 2^20 code points to the 2^16 already available (this leads to the surprising upper limit of 0x10FFFF).

In parallel, UTF-8 was created as an ASCII-compatible form of Unicode representation that would become the dominant form of text in the Web and other interchange environments.

The UCS-2 based character models of the legacy 16-bit platforms in many cases couldn't be updated for fully embracing UTF-16 right away. For instance, only much later did ECMAScript introduce the "u" (Unicode) flag for regular expressions to have them actually match "Unicode" characters. So, on these platforms, UTF-16 is handled in a UCS-2 character model, and sometimes orphaned surrogates leak out instead of Unicode characters as "code points" in interfaces that are not meant to impose these implementation limitations on the outside world.

UTF-8 is unaffected by this UTF-16 quirk and of course doesn't support encoding surrogates. (UTF-8 is careful to allow a single representation only for each Unicode character, and enabling the alternative use of surrogate pairs would violate that invariant, while isolated surrogates don't mean anything in Unicode.)

Newly designed IETF protocols typically do not have to consider these problems, but occasionally there are attempts to include isolated surrogate code points into what we call Unicode characters here.

Appendix C. Normalization

Please see Section 3 of [RFC5198] for a brief introduction to Unicode normalization and Normalization Form C (NFC). However, since that section was written, additional experience with normalization has led to the realization that the Unicode normalization rules do not always preserve certain details in certain writing systems.

Therefore, the implementation approach of routinely normalizing all text before interchange has fallen out of favor. Instead, implementations are encouraged to use text sources that already generally use NFC except where normalization would have been harmful. Where two text strings needs to be compared, it may be appropriate to apply normalization to both text strings for the purposes of the comparison only.

Additional complications can come from the fact that some implementations of applications may rely on operating system libraries over which they have little control. The need to maintain interoperability in such environments suggests that receivers of MNU

should be prepared to receive unnormalized text and should not react to that in excessive ways; however, there also is no expectation for receivers to go out of their way doing so.

Appendix D. Relationship to RFC 5198

Of the mandates listed in Section 2, the third and fourth requirement are also posed by [RFC5198], while the first two remove further legacy compatibility considerations.

[RFC5198] contains some discussion and background material that the present document does not attempt to repeat; the interested reader may therefore want to consult it as an informative reference.

Mandates of [RFC5198] that are specific to a version of Unicode are not picked up in this specification, e.g., there is no check for unassigned code points. Some implementations may want to add such a check; however, in general, this can hinder further evolution as it may become hard to use new characters as long as not every component on the way has been upgraded. (See also Section 5.8.)

D.1. Going beyond RFC 5198

The handling of line endings (with 2D MNU prescribing LF as the line terminator, and adding or specifying CRLF line endings as variances) may be controversial. In particular, calling out CR-tolerance as an extra (and often undesirable) feature may seem novel to some readers. The handling as specified here is much closer to the way line endings are handled on the software side than the cumbersome rules of [RFC5198]. More generally speaking, one could say that the present specification is intended to be used by state-of-the-art protocols going forward, maybe less so by existing protocols.

Even in the “with CR-tolerant lines” variance, the CR character is only allowed as an embellishment of an immediately following LF character. This reflects the fact that overprinting has only seen niche usage for quite a number of decades now (and otherwise has been supplanted by the concept of “combining characters”).

Unicode Line and Paragraph separators probably seemed like a good idea at the time, but have not taken hold. Today, their occurrence is more likely to trigger a bug or even serve as an attack.

HT characters (“TABs”) were needed on ASR33 terminals to speed up processing of blank spaces at 110 bit/s line speed. Unless some legacy applications require compatibility with this ancient and frequently varied convention, HT characters are no longer appropriate in Modern Network Unicode. In support of legacy compatibility cases that do require tolerating their use, the “with HT characters” variance is defined.

The version-nonspecific nature of MNU creates some fuzziness that may be undesirable but is more realistic in environments where applications choose the Unicode version with the Unicode library that happens to be available to them.

D.2. Byte order marks

For UTF-8, there is no encoding ambiguity and thus no need for a byte order mark. However, some systems have made regular use of a leading U+FEFF character in UTF-8 files, anyway, often in order to mark the file as UTF-8 in case other character codings are also in use and metadata is not available. This destroys the ASCII compatibility of UTF-8; it also creates problems when systems then start to expect a BOM in UTF-8 input and none is provided. Section 6 of RFC 3629 [STD63] also RECOMMENDS not using Byte Order Marks with UTF-8 when it is clear that this charset is being used, but does not phrase this as an unambiguous mandate, so we add that here (as did [RFC5198]), unless permitted by a variance.

Some background on the construct of byte order marks: The 16-bit and 32-bit encodings for Unicode are available in multiple byte orders. The byte order in use in a specific piece of text can be provided by metadata (such as a media type) or by prefixing the text with a “Byte Order Mark”, U+FEFF. Since code point U+FFFE is never used in Unicode, this unambiguously identifies the byte order. There is no need to indicate a byte order in UTF-8; this discussion only relates to the secondary use of the character used in byte order marks as a UTF-8 signature in otherwise unspecified text files.

Acknowledgements

Klaus Hartke and Henk Birkholz drove the author out of his mind enough to make him finally write this up. James Manger, Tim Bray and Martin Thomson provided comments on an early version of this draft. Doug Ewell proposed to define an ABNF rule NONASCII, of which we have included the essence.

Author’s Address

Carsten Bormann
Universitt Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org