Network Working Group
Internet-Draft
Updates: 5234 (if approved)
Intended Status: Experimental
Expires: May 4, 2017

S. Leonard
Penango, Inc.
C. Newman
Oracle

October 31, 2016

Unicode in ABNF
draft-seantek-unicode-in-abnf-02

Abstract

This experimental document adds support for Unicode strings in ABNF (Augmented Backus-Naur Form), and provides certain symbols related to Unicode code point ranges.

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 http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft is a fork of draft-seantek-abnf-more-core-rules-05.

1. Introduction

Augmented Backus-Naur Form (ABNF) [RFC5234] is a formal syntax that is popular among many Internet specifications. Many Internet documents employ this syntax along with the Core Rules defined in Appendix B.1 of [RFC5234]. ABNF is defined in terms of ASCII [ASCII86, RFC0020]; however, Unicode [UNICODE] has become increasingly popular--even required--as the Internet has evolved over the last two decades. Unicode (as UTF-8) will be permitted in the RFC series [IABNA], while [RFC5198] established Net-Unicode as the standard form for the use of Unicode as "network text". Protocols that originally were ASCII-based have been, or are being, extended to support Unicode. However, protocols that use Unicode in some way (e.g., permit UTF-8 content in a production) use different ABNF expressions, some of which do not conform to the modern Unicode Standard 9.0.0, and therefore could introduce interoperability or security problems.

Many parties have expressed interest in incorporating [UNICODE] into ABNF, yet the questions remain: "How?" and "To what extent?"

This document proposes standardized techniques for expressing Unicode code points using ABNF. This document intends to be very conservative in its approach: a conforming implementation only needs to know how to map between the Unicode scalar values and any Unicode encoding form. The Unicode Character Database (UCD, Section 4.1 of [UNICODE]) is intentionally not necessary. ABNF text that uses the syntax in this document needs to be in a Unicode encoding form (Conformance Clause D89 of [UNICODE]), but ABNF text that just uses the rules or terminal values can be expressed in ASCII [RFC0020].

2. Unicode Code Points in ABNF

(Consult Section 2.3 of [RFC5234] in relation to this paragraph.) Unicode has been expressed in several different ways in RFCs to-date. This document establishes that in contexts where Unicode is specified as the coded character set [RFC2130], the terminal values %x00-10FFFF are to be used to represent the Unicode code points. Only the Unicode scalar values are to be used in specifications that follow this document; surrogate code points (%xD800-DFFF) are not to be used [[NB: directly]]. This technique aligns ABNF with W3C EBNF [XMLEBNF] and Unicode EBNF [UNICODE].

(Consult Section 2.4 and Appendix B.2 of [RFC5234] in relation to this paragraph.)
In contexts where Unicode is specified as the character set, the ABNF-based grammar may have multiple external encodings. This document does not fix the encoding scheme. The obvious external

encoding is UTF-8 (see Net-Unicode [RFC5198]), but other encodings are possible. This document neither restricts productions to NFC, nor provides a syntax for normalization to NFC.

3. Unicode Core Rule Update

Appendix A furnishes Unicode Core Rules that include comprehensive support for certain Unicode ranges and characters. These Unicode Core Rules supplement the Core Rules of [RFC5234] and [ABNFMORE]; they are intended to be available whenever this document is invoked.

The rules reflect broad categories of allowable and disallowable characters in protocols for interchange between systems, as the Internet community has evolved, and as of Unicode 9.0.0 in August 2016 [UNICODE]. It is a design goal that a general-purpose ABNF grammar should not need to delve into the minutiae of Unicode character properties, which can be tailorable (i.e., language-specific), overridable, and unstable (between Unicode versions). It is a further design goal that a general-purpose ABNF grammar should not need to rely on sizeable external sources, namely the Unicode Character Database (Section 4.1 of [UNICODE]). To constrain this document's scope, character properties are not addressed further.

According to a survey of all RFCs published through August 2016, many widely used Internet protocols rely on horizontal whitespace (HT and SP, or occasionally SP alone) and line breaks (usually CRLF, sometimes LF) as delimiters. Therefore, the rules specifically address horizontal whitespace and line breaks.

Rules that both include and exclude the private-use characters (Section 23.5 of [UNICODE]) are provided. Private-use characters "are intended for open interchange, subject to interpretation by private agreement" (Section 23.7 of [UNICODE]). Therefore, there is no way within [UNICODE] itself to provide for a common interpretation of these code points. See also Section 4 of [RFC5198]. A protocol designer needs to establish that common interpretation in prose,

provide for protocol elements that establish the common interpretation, or (explicitly) accept that a common interpretation is done outside of the designer's protocol.

4. Case-Sensitive Unicode String Syntax

This document extends ABNF with a new case-sensitive Unicode string literal. The type is denoted using a type prefix similar to the type prefixes used with numeric values and case-sensitive ASCII string literals. No syntax is provided for a case-insensitive Unicode string literal because doing so would require implementing Unicode caseless matching [UNICODE], which is language-dependent, Unicode version-

dependent, and very complicated overall. Caseless matching also requires the UCD.

Add the contents of Section 4.1 to [RFC5234].

4.1. Terminal Values - Literal Text Strings

Literal case sensitive text strings in ABNF may be in the Unicode character set [UNICODE]. The following prefix is used:

```
%su          =  case-sensitive, Unicode
```

To be consistent with prior implementations of ABNF, having no prefix means that the string is case insensitive and in ASCII.

[[ALT/DISCUSS: [RFC7405] %s"text" could be extended to support characters beyond ASCII. It is a strict superset of [RFC7405] and thus simpler. This document would leave [%i]"text" undefined for the time being, or, a collation from [RFC4790] could be identified.]]

The case-sensitive Unicode string can be comprised of any Graphic, Format, or Reserved code point. Control, Private-Use, Surrogate, and Noncharacter code points are excluded. Newline (line breaking) characters are also omitted. (See Table 2-3 of [UNICODE].)

An example:

```
rulename = %su"!100Q$"
```

where the character ! is actually the Unicode code point U+00A5 YEN SIGN, and the character $ is actually the Unicode code point U+1F39F ADMISSION TICKETS, is equivalent to the rule:

```
rulename = %xA5.31.30.30.51.1F39F
```

4.2. ABNF Definition of ABNF - char-val

```
char-val      =/  case-sensitive-Unicode-string

; ALT/DISCUSS: "%s", modify 7405
case-sensitive-Unicode-string =
                  "%su" quoted-Unicode-string

quoted-Unicode-string = DQUOTE *(%x20-21 / %x23-7E /
                        UVCHARBEYONDASCII) DQUOTE
                          ; quoted string of SP and VCHAR
                          ;  without DQUOTE, and UVCHAR
                          ;  beyond the ASCII range
```


5. Terminal Value Transformation Syntax for UTF-8 and UTF-16

While Section 2 establishes terminal values %x00-10FFFF for Unicode, many Internet protocols incorporate Unicode using UTF-8 and define protocol elements using UTF-8 terminal values (i.e., values in the 8-bit range of %x00-FF, or more specifically, %x00-BF and %xC2-F4); see [RFC3629]. A smaller yet notable set of protocols use UTF-16.

Writing out Unicode code points or ranges in UTF-8 or UTF-16 can be cumbersome and error-prone. This document therefore provides a "terminal value transformation syntax", so that the code points %x00-10FFFF can be written out natively, but the resulting ABNF represents 8-bit or 16-bit units at the level of ABNF syntax. From there, a protocol can supply a specific mapping (encoding) of those values into a character set or other representation, consistent with Section 2.3 of [RFC5234].

The syntax is:

```
 %t8(...)    for 8-bit UTF-8 (transform to %x00-BF and %xC2-F4)
 %t16(...)   for 16-bit UTF-16 (transform to %x00-D7FF,
             %xD800-DBFF %xDC00-DFFF, and %xE000-FFFF)
 %t16le(...) for 8-bit UTF-16LE (transform to %x00.00-%xFF.FF,
```

```
              little-endian)
 %t16be(...) for 8-bit UTF-16BE (transform to %x00.00-%xFF.FF,
              big-endian)
```

[[NB: Other possibilities: !t8 ~t8 $t8 #t8 -t8]]

A transform is applied by recursively driving it into the elements, transforming terminal values from the original code point to the corresponding Unicode Transformation Format over an 8-bit (or 16-bit) field. The transforms in this document distribute over ABNF operators. "%t16" outputs 16-bit terminal values from %x00-FFFF, meaning that the endianness is not specified: a protocol needs to specify this or furnish a protocol slot for 16-bit code units. In contrast, "%t16be" and "%t16le" output 8-bit terminal values: each terminal value in the input will correspond to two or four terminal values in the output.

If a transform is used on a terminal value outside the Unicode scalar value range (see the proposed Core Rule <UNICODE>), the resulting terminal value can be neither satisfied nor produced.

A "reverse transformation syntax" to go from 8-bit or 16-bit terminal values to reassembled Unicode code points is not proposed at this time.

5.1. Examples

Example 1: The following rules are equivalent; see [RFC3629]:

```
 UTF8-MB     = UTF8-2 / UTF8-3 / UTF8-4  ; from RFC 3629

                     ; %x80-D7FF / %xE000-10FFFF
 UTF8-MB     = %t8( BEYONDASCII )
```

Example 2: The code point U+1F430 RABBIT FACE can be represented as %x1F430. It can also be represented as %xD83D.DC30 or %t16(%x1F430) when UTF-16 is intended.

5.2. Advantages and Features

Using transformation syntax offers several advantages:

The generic ABNF syntax of a textual protocol can take full advantage of the Unicode character set; the syntax is not dependent on a particular encoding form.

Specifying ranges of characters becomes unwieldy when explicitly defined in terms of code units in a Unicode encoding form, e.g., as UTF-8 code units (octets) for characters beyond ASCII, or as UTF-16 code units (16-bit words) for supplementary characters. Trying to specify Punycode in ABNF would be, for all intents and purposes, impossible! (Note: it's not actually impossible, but very difficult and not particularly useful.)

Protocols that have arbitrary binary slots (e.g., BINARYMIME) are inherently incompatible with Section 2 syntax, but compatibility can be achieved by using transformation syntax.

Protocol designers can effectively exploit the "holes" in UTF-8, because octets C0, C1, and F5-FF are never seen in UTF-8. These octets provide natural delimiters for arbitrary runs of UTF-8. An advantage of using such octets as delimiters is that checking for these octets has to be done anyway for security reasons, so a designer can save cycles by incorporating this part of a check for well-formed Unicode into a protocol. Such delimiters can only be expressed outside of "%t8", since a "%t8" transform will never produce those terminal values.

(UTF-16 also has such "holes", namely, in unpaired surrogates. But using unpaired surrogates as delimiters may suffer from other security pitfalls; in any event, UTF-16 is far less common in IETF usage.)

6. Comment Syntax

This document extends ABNF to have Unicode comments. Comments are treated as specification prose, so they may be normative depending on the context. Comment text allows for the same repertoire of characters as RFC text. The RFC Editors can regulate comments to the same extent as specification prose, including disallowing certain characters or code points.

6.1. Comment: ; Comment

(No changes to the text of Section 3.9 of [RFC5234] are needed.)

6.2. ABNF Definition of ABNF - comment

```
; given:
comment      =    ";" *(WSP / VCHAR) CRLF

; increment (unambiguous grammar):
comment      =/   ";" *(UWSP / UVCHAR / PUACHAR)
                  (UWSPBEYONDASCII / UVCHARBEYONDASCII / PUACHAR)
                       *(UWSP / UVCHAR / PUACHAR) CRLF

; or redefine:
comment      =    ";" *(UWSP / UVCHAR / PUACHAR) CRLF
```

7. Notational Conventions

For readability it is advisable to express a Unicode code point as the character itself, the numeric terminal value, and the name or a name alias. Only one expression is used for the formal ABNF notation: either the character itself (Section 4) or the numeric terminal value (Section 2). The other expressions can be incorporated into an adjacent comment.

The suggested notational convention for the adjacent comment follows Appendix A of [UNICODE]. The comment text is comprised of one or more WSP characters, optionally either the character itself or "U+" syntax followed by exactly one SP, and the name or a name alias in ALL-CAPS ASCII. Multiple characters can be notated in sequence on multiple comment lines or on a single comment line. It is neither advisable nor necessary to notate characters in the ASCII range. Examples of the notation include:


```
; U+2206 INCREMENT
```

```
; U+2030 PER MILLE SIGN
change-in-temp = %su"$" 3DIGIT %su"%"

;       # EURO SIGN    ZWJ    / VULGAR FRACTION ONE HALF
euros = %x20AC 3DIGIT [%x200D.BD]
```

where the characters $, %, #, and / are actually the respective Unicode characters mentioned in the comments.

8. Effects on RFC 5234

Formally, this document updates [RFC5234] but does not modify it in situ. Authors need to reference this document if they want to include these enhancements; bare references to [RFC5234] do not include this specification (or, for that matter, [RFC7405]). This directive follows a model whereby document authors can choose whether to invoke particular enhancements to ABNF. As time goes on, the IETF can determine how often these enhancements are invoked, and can decide whether to include them as part of a revision to the base [RFC5234].

A bare reference to this document invokes the case-sensitive Unicode literal string syntax enhancement, the Unicode comment syntax enhancement, and the Unicode Core Rules of Appendix A (i.e., the Core Rules do not have to be further referenced). Nevertheless, document authors are free to qualify a reference to this document to invoke each feature selectively.

Appendix A of this document is meant to supplement Appendix B.1 of [RFC5234] and Appendix A of [ABNFMORE]; therefore, concurrently referencing those documents is a good idea. Document authors who reference this document should use the rules of Appendix A, and should not attempt to redefine or provide incremental alternatives to them (except for backwards compatibility with prior documents).

9. IANA Considerations

This document implies no IANA considerations.

10. Security Considerations

While the Unicode Core Rules themselves may not be security-relevant, the use of C1 control characters could very well be security-relevant, because they may trigger special functions on various devices, while being invisible in other contexts. Similarly, case-sensitive Unicode string syntax allows for a broad range of code points, many of which represent characters that are confusable with other characters, or can only be inferred by visible yet subtle

changes in the surrounding graphemes (or worse, semantic changes that do not have visual representations).

Protocols using Unicode should evaluate the applicability of Unicode security considerations [UTR#36].

11. References

11.1. Normative References

[ASCII86] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[UNICODE] The Unicode Consortium, "The Unicode Standard, Version 9.0.0", The Unicode Consortium, August 2016.

11.2. Informative References

[IABNA] Flanagan, H., "The Use of Non-ASCII Characters in RFCs", draft-iab-rfc-nonascii-02 (work in progress), April 2016.

[RFC1345] Simonsen, K., "Character Mnemonics and Character Sets", RFC 1345, June 1992.

[RFC2130] Weider, C., Preston, C., Simonsen, K., Alvestrand, H., Atkinson, R., Crispin, M., and P. Svanberg, "The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996", RFC 2130, April 1997.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, March 2007.

7405, December 2014.

[UTR#36] Davis, M. and M. Suignard, "Unicode Security Considerations", Unicode Technical Report #36, September 2014, <http://unicode.org/reports/tr36/>.

[XMLEBNF] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", Section 6, W3C Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

Appendix A. Comprehensive Unicode Core Rules

Certain basic rules are in uppercase, such as SP, HTAB, CRLF, DIGIT, ALPHA, etc.

```
; D76 Unicode scalar value

UNICODE      = <U+0000-U+D7FF / U+E000-U+10FFFF>
BEYONDASCII  = <U+0080-U+D7FF / U+E000-U+10FFFF>
BEYONDG0     = <U+0080-U+D7FF / U+E000-U+10FFFF>

C1           = <U+0080-U+009F>
BEYONDC1     = <U+00A0-U+D7FF / U+E000-U+10FFFF>
G1           = <U+00A0-U+00FF>  ; 96-set
BEYONDG1     = <U+0100-U+D7FF / U+E000-U+10FFFF>
LATIN1       = <U+0000-U+00FF>
BEYONDLATIN1 = <U+0100-U+D7FF / U+E000-U+10FFFF>

; C2 D14 noncharacter (sentinel)
; Section 23.7 Noncharacters, see also NUL

NONUCHAR     = <U+FDD0-U+FDEF / U+FFFE-U+FFFF /
                U+1FFFE-U+1FFFF / U+2FFFE-U+2FFFF /
                U+3FFFE-U+1FFFF / U+4FFFE-U+4FFFF /
                U+5FFFE-U+1FFFF / U+6FFFE-U+6FFFF /
                U+7FFFE-U+1FFFF / U+8FFFE-U+8FFFF /
```

```
                    U+9FFFE-U+1FFFF / U+AFFFE-U+AFFFF /
                    U+BFFFE-U+1FFFF / U+CFFFE-U+CFFFF /
                    U+DFFFE-U+1FFFF / U+EFFFE-U+EFFFF /
                    U+FFFFE-U+FFFFF / U+10FFFE-U+10FFFF>

    ; UCHAR rules are analogous to CHAR

    UCHARBEYONDBMP = <U+10000-U+1FFFD / U+20000-U+2FFFD /
                     U+30000-U+3FFFD / U+40000-U+4FFFD /
```



```
                    U+50000-U+5FFFD / U+60000-U+6FFFD /
                    U+70000-U+7FFFD / U+80000-U+8FFFD /
                    U+90000-U+9FFFD / U+A0000-U+AFFFD /
                    U+B0000-U+BFFFD / U+C0000-U+CFFFD /
                    U+D0000-U+DFFFD / U+E0000-U+EFFFD /
                    U+F0000-U+FFFFD / U+100000-U+10FFFD>

    UCHARBEYONDLATIN1 = <U+0100-U+D7FF / U+E000-U+FDCF /
                         U+FDF0-U+FFFD> / UCHARBEYONDBMP

    UCHARBEYONDC1  = <U+00A0-U+D7FF / U+E000-U+FDCF / U+FDF0-U+FFFD>
                      / UCHARBEYONDBMP

    UCHARBEYONDASCII = C1 / UCHARBEYONDC1

    UCHAR        = <U+0001-U+D7FF / U+E000-U+FDCF / U+FDF0-U+FFFD> /
                    UCHARBEYONDBMP

    ; D49 private-use
    ; Section 23.5 Private-Use Characters

    ; Primary Private Use Area (in BMP)
    PPUACHAR     = <U+E000-U+F8FF>
    ; Supplementary Private Use Area-A
    SPUAACHAR    = <U+F0000-U+FFFFF>
    ; Supplementary Private Use Area-B
    SPUABCHAR    = <U+100000-U+10FFFF>

    ; TODO: possible alternates: PUCHAR, PUA
    PUACHAR        = PPUACHAR / SPUAACHAR / SPUABCHAR

    ; Unicode-y VCHAR: like VCHAR, attempts to capture
```

```
; "all standardized graphic and formatting
; characters/code points for open interchange,
; excluding white space and controls"
; EXCLUDES: Noncharacters (some Cn), Cs, Co, Cc, Z (Zs, Zl, Zp)

UVCHARBEYONDBMP = <U+10000-U+1FFFD / U+20000-U+2FFFD /
                   U+30000-U+3FFFD / U+40000-U+4FFFD /
                   U+50000-U+5FFFD / U+60000-U+6FFFD /
                   U+70000-U+7FFFD / U+80000-U+8FFFD /
                   U+90000-U+9FFFD / U+A0000-U+AFFFD /
                   U+B0000-U+BFFFD / U+C0000-U+CFFFD /
                   U+D0000-U+DFFFD / U+E0000-U+EFFFD>

UVCHARBEYONDLATIN1 = <U+0100-U+167F / U+1681-U+1FFF /
                      U+200B-U+2027 / U+202A-U+202E /
                      U+2030-U+205E / U+2060-U+2FFF /
```



```
                      U+3001-U+D7FF /
                      U+F900-U+FDCF / U+FDF0-U+FFFD> /
                     UVCHARBEYONDBMP

UVCHARBEYONDASCII = <U+00A1-U+167F / U+1681-U+1FFF /
                     U+200B-U+2027 / U+202A-U+202E /
                     U+2030-U+205E / U+2060-U+2FFF /
                     U+3001-U+D7FF /
                     U+F900-U+FDCF / U+FDF0-U+FFFD> /
                    UVCHARBEYONDBMP

UVCHARBEYONDC1 = UVCHARBEYONDASCII

UVCHAR       =  VCHAR / UVCHARBEYONDASCII

; horizontal white space only (Zs beyond ASCII),
; NO line breaks (Cc, Zl, Zp)
; cf Section 5.8 Newline Guidelines with RFC 5198
; see also SP
UWSPBEYONDASCII = <U+00A0 / U+1680 / U+2000-U+200A /
                   U+202F / U+205F / U+3000>

; includes HT
UWSP         = WSP / UWSPBEYONDASCII
```

```
; C1 Controls
PAD          = <U+0080> ; gov't health warning: figment
HOP          = <U+0081> ; gov't health warning: figment
BPH          = <U+0082>
NBH          = <U+0083>
IND          = <U+0084>
NEL          = <U+0085>
; NLF          CRLF, CR, LF, NEL (not LS or PS)
;  --probably unnecessary for Internet usage:
;  CRLF is already the standard
SSA          = <U+0086>
ESA          = <U+0087>
HTS          = <U+0088>
HTJ          = <U+0089>
VTS          = <U+008A>
PLD          = <U+008B>
PLU          = <U+008C>
RI           = <U+008D>
SS2          = <U+008E>
SS3          = <U+008F>
DCS          = <U+0090>
PU1          = <U+0091>
PU2          = <U+0092>
```



```
STS          = <U+0093>
CCH          = <U+0094>
MW           = <U+0095>
SPA          = <U+0096>
EPA          = <U+0097>
SOS          = <U+0098>
SGCI         = <U+0099> ; or SGC, gov't health warning: figment
SCI          = <U+009A>
CSI          = <U+009B>
ST           = <U+009C>
OSC          = <U+009D>
PM           = <U+009E>
APC          = <U+009F>

; Latin1
NBSP         = <U+00A0>
SHY          = <U+00AD>
```

```
      ; Zl, Zp
      ; NB: These are excluded from both UVCHAR and UWSP
      LS            = <U+2028>
      PS            = <U+2029>
```

Authors' Addresses

Sean Leonard
Penango, Inc.
5900 Wilshire Boulevard
21st Floor
Los Angeles, CA 90036
USA

EMail: dev+ietf@seantek.com
URI: http://www.penango.com/

Chris Newman
Oracle
440 E. Huntington Dr., Suite 400
Arcadia, CA 91006
USA

EMail: chris.newman@oracle.com