Stream: Internet Engineering Task Force (IETF)
RFC: 9682
Updates: 8610
Category: Standards Track
Published: November 2024
ISSN: 2070-1721
Author: C. Bormann
Universität Bremen TZI

RFC 9682
Updates to the Concise Data Definition Language (CDDL) Grammar

Abstract

The Concise Data Definition Language (CDDL), as defined in RFCs 8610 and 9165, provides an easy and unambiguous way to express structures for protocol messages and data formats that are represented in Concise Binary Object Representation (CBOR) or JSON.

This document updates RFC 8610 by addressing related errata reports and making other small fixes for the ABNF grammar defined for CDDL.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9682.

Copyright Notice

Table of Contents

1. Introduction

The Concise Data Definition Language (CDDL), as defined in [RFC8610] and [RFC9165], provides an easy and unambiguous way to express structures for protocol messages and data formats that are represented in CBOR or JSON.

This document updates [RFC8610] by addressing errata reports and making other small fixes for the ABNF grammar defined for CDDL. The body of this document explains and shows motivation for the updates; the updated collected ABNF syntax in Figure 11 in Appendix A replaces the collected ABNF syntax in Appendix B of [RFC8610].

1.1. Conventions and Definitions

The terminology from [RFC8610] applies. The grammar in [RFC8610] is based on ABNF, which is defined in [STD68] and [RFC7405].

2. Clarifications and Changes Based on Errata Reports

A number of errata reports have been made regarding some details of text string and byte string literal syntax: for example, [Err6527] and [Err6543]. These are being addressed in this section, updating details of the ABNF for these literal syntaxes. Also, the changes described in [Err6526] need to be applied (backslashes have been lost during the RFC publication process of Appendix G.2 of [RFC8610], garbling the text explaining backslash escaping).

These changes are intended to mirror the way existing implementations have dealt with the errata reports. This document also uses the opportunity presented by the necessary cleanup of the grammar of string literals for a backward-compatible addition to the syntax for hexadecimal escapes. The latter change is not automatically forward compatible (i.e., CDDL specifications that make use of this syntax do not necessarily work with existing implementations until these are updated, which is recommended by this specification).

2.1. Updates to String Literal Grammar

2.1.1. Erratum ID 6527 (Text String Literals)

The ABNF used in [RFC8610] for the content of text string literals is rather permissive:

```
; ABNF from RFC 8610:
text = %x22 *SCHAR %x22
SCHAR = %x20-21 / %x23-5B / %x5D-7E / %x80-10FFFD / SESC
SESC = "\" (%x20-7E / %x80-10FFFD)
```

Figure 1: Original ABNF from RFC 8610 for Strings with Permissive ABNF for SESC (Which Did Not Allow Hex Escapes)

This allows almost any non-C0 character to be escaped by a backslash, but critically misses out on the `\uXXXX` and `\uHHHH\uLLLL` forms that JSON allows to specify characters in hex (which should apply here according to item 6 of Section 3.1 of [RFC8610]). (Note that CDDL imports from JSON the unwieldy `\uHHHH\uLLLL` syntax, which represents Unicode code points beyond U+FFFF by making them look like UTF-16 surrogate pairs; CDDL text strings do not use UTF-16 or surrogates.)

Both can be solved by updating the SESC rule. This document uses the opportunity to add a popular form of directly specifying characters in strings using hexadecimal escape sequences of the form `\u{hex}`, where `hex` is the hexadecimal representation of the Unicode scalar value. The result is the new set of rules defining SESC in Figure 2.

```
; new rules collectively defining SESC:
SESC = "\" ( %x22 / "/" / "\" /                 ; \" \/ \\
            %x62 / %x66 / %x6E / %x72 / %x74 / ; \b \f \n \r \t
            (%x75 hexchar) )                    ; \uXXXX
hexchar = "{" (1*"0" [ hexscalar ] / hexscalar) "}" /
          non-surrogate / (high-surrogate "\" %x75 low-surrogate)
non-surrogate = ((DIGIT / "A"/"B"/"C" / "E"/"F") 3HEXDIG) /
                ("D" %x30-37 2HEXDIG )
high-surrogate = "D" ("8"/"9"/"A"/"B") 2HEXDIG
low-surrogate = "D" ("C"/"D"/"E"/"F") 2HEXDIG
hexscalar = "10" 4HEXDIG / HEXDIG1 4HEXDIG
          / non-surrogate / 1*3HEXDIG
HEXDIG1 = DIGIT1 / "A" / "B" / "C" / "D" / "E" / "F"
```

Figure 2: Update to String ABNF in Appendix B of [RFC8610]: Allow Hex Escapes

> Notes: In ABNF, strings such as `"A"`, `"B"`, etc., are case insensitive, as is intended here. The rules above could have also used `%s"b"`, etc., instead of `%x62`, but didn't, in order to maximize compatibility with ABNF tools.

Now that SESC is more restrictively formulated, an update to the BCHAR rule used in the ABNF syntax for byte string literals is also required:

```
; ABNF from RFC 8610:
bytes = [bsqual] %x27 *BCHAR %x27
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
bsqual = "h" / "b64"
```

Figure 3: ABNF from RFC 8610 for BCHAR

With the SESC updated as above, `\'` is no longer allowed in BCHAR and now needs to be explicitly included there; see Figure 4.

2.1.2. Erratum ID 6278 (Consistent String Literals)

Updating BCHAR also provides an opportunity to address [Err6278], which points to an inconsistency in treating U+007F (DEL) between SCHAR and BCHAR. As U+007F is not printable, including it in a byte string literal is as confusing as for a text string literal; therefore, it should be excluded from BCHAR as it is from SCHAR. The same reasoning also applies to the C1 control characters, so the updated ABNF actually excludes the entire range from U+007F to U+009F. The same reasoning also applies to text in comments (PCHAR). For completeness, all these rules should also explicitly exclude the code points that have been set aside for UTF-16 surrogates.

```
; new rules for SCHAR, BCHAR, and PCHAR:
SCHAR = %x20-21 / %x23-5B / %x5D-7E / NONASCII / SESC
BCHAR = %x20-26 / %x28-5B / %x5D-7E / NONASCII / SESC / "\'" / CRLF
PCHAR = %x20-7E / NONASCII
NONASCII = %xA0-D7FF / %xE000-10FFFD
```

Figure 4: Update to ABNF in Appendix B of [RFC8610]: BCHAR, SCHAR, and PCHAR

(Note that, apart from addressing the inconsistencies, there is no attempt to further exclude non-printable characters from the ABNF; doing this properly would draw in complexity from the ongoing evolution of the Unicode standard [UNICODE] that is not needed here.)

2.1.3. Addressing Erratum ID 6526 and Erratum ID 6543

The above changes also cover [Err6543] (a proposal to split off qualified byte string literals from UTF-8 byte string literals) and [Err6526] (lost backslashes); see Appendix B for details.

2.2. Examples Demonstrating the Updated String Syntaxes

The CDDL example in Figure 5 demonstrates various escaping techniques now available for (byte and text) strings in CDDL. Obviously, in the literals for `a` and `x`, there is no need to escape the second character, an `o`, as `\u{6f}`; this is just for demonstration. Similarly, as shown in `c` and `z`, there also is no need to escape the "🁳" (DOMINO TILE VERTICAL-02-02, U+1F073) or "⌘" (PLACE OF INTEREST SIGN, U+2318); however, escaping them may be convenient in order to limit the character repertoire of a CDDL file itself to ASCII [STD80].

```
start = [a, b, c, x, y, z]

; "🁢", DOMINO TILE VERTICAL-02-02, and
; "⌘", PLACE OF INTEREST SIGN, in a text string:
a = "D\u{6f}mino's \u{1F073} + \u{2318}"      ; \u{}-escape 3 chars
b = "Domino's \uD83C\uDC73 + \u2318"          ; escape JSON-like
c = "Domino's 🁢 + ⌘"                         ; unescaped

; in a byte string given as text, the ' needs to be escaped:
x = 'D\u{6f}mino\u{27}s \u{1F073} + \u{2318}' ; \u{}-escape 4 chars
y = 'Domino\'s \uD83C\uDC73 + \u2318'         ; escape JSON-like
z = 'Domino\'s 🁢 + ⌘'                        ; escape ' only
```

Figure 5: Example Text and Byte String Literals with Various Escaping Techniques

In this example, the rules a to c and x to z all produce strings with byte-wise identical content: a to c are text strings and x to z are byte strings. Figure 6 illustrates this by showing the output generated from the `start` rule in Figure 5, using pretty-printed hexadecimal.

```
86                                      # array(6)
   73                                   # text(19)
      446f6d696e6f277320f09f81b3202b20e28c98 # "Domino's 🁢 + ⌘"
   73                                   # text(19)
      446f6d696e6f277320f09f81b3202b20e28c98 # "Domino's 🁢 + ⌘"
   73                                   # text(19)
      446f6d696e6f277320f09f81b3202b20e28c98 # "Domino's 🁢 + ⌘"
   53                                   # bytes(19)
      446f6d696e6f277320f09f81b3202b20e28c98 # "Domino's 🁢 + ⌘"
   53                                   # bytes(19)
      446f6d696e6f277320f09f81b3202b20e28c98 # "Domino's 🁢 + ⌘"
   53                                   # bytes(19)
      446f6d696e6f277320f09f81b3202b20e28c98 # "Domino's 🁢 + ⌘"
```

Figure 6: Generated CBOR from CDDL Example (Pretty-Printed Hexadecimal)

3. Small Enabling Grammar Changes

Each subsection that follows specifies a small change to the grammar that is intended to enable certain kinds of specifications. These changes are backward compatible (i.e., CDDL files that comply with [RFC8610] continue to match the updated grammar) but not necessarily forward compatible (i.e., CDDL specifications that make use of these changes cannot necessarily be processed by existing implementations of [RFC8610]).

3.1. Empty Data Models

[RFC8610] requires a CDDL file to have at least one rule.

```
; ABNF from RFC 8610:
cddl = S 1*(rule S)
```

Figure 7: ABNF from RFC 8610 for Top-Level Rule `cddl`

This makes sense when the file has to stand alone, as a CDDL data model needs to have at least one rule to provide an entry point (i.e., a start rule).

With CDDL modules [CDDL-MODULES], CDDL files can also include directives, and these might be the source of all the rules that ultimately make up the module created by the file. Any other rule content in the file has to be available for directive processing, making the requirement for at least one rule cumbersome.

Therefore, the present update extends the grammar as in Figure 8 and turns the existence of at least one rule into a semantic constraint, to be fulfilled after processing of all directives.

```
; new top-level rule:
cddl = S *(rule S)
```

Figure 8: Update to Top-Level ABNF in Appendices B and C of RFC 8610

3.2. Non-Literal Tag Numbers and Simple Values

The existing ABNF syntax for expressing tags in CDDL is as follows:

```
; extracted from the ABNF in RFC 8610:
type2 =/ "#" "6" ["." uint] "(" S type S ")"
```

Figure 9: Original ABNF from RFC 8610 for Tag Syntax

This means tag numbers can only be given as literal numbers (uints). Some specifications operate on ranges of tag numbers; for example, [RFC9277] has a range of tag numbers 1668546817 (0x63740101) to 1668612095 (0x6374FFFF) to tag specific content formats. This cannot currently be expressed in CDDL. Similar considerations apply to simple values (`#7.xx`).

This update extends the syntax to the following:

```
; new rules collectively defining the tagged case:
type2 =/ "#" "6" ["." head-number] "(" S type S ")"
       / "#" "7" ["." head-number]
head-number = uint / ("<" type ">")
```

Figure 10: Update to Tag and Simple Value ABNF in Appendices B and C of RFC 8610

For `#6`, the `head-number` stands for the tag number. For #7, the `head-number` stands for the simple value if it is in the ranges 0..23 or 32..255 (as per Section 3.3 of RFC 8949 [STD94], the simple values 24..31 are not used). For 24..31, the `head-number` stands for the "additional information", e.g., `#7.25` or `#7.<25>` is a float16, etc. (All ranges mentioned here are inclusive.)

So the above range can be expressed in a CDDL fragment such as:

```
ct-tag<content> = #6.<ct-tag-number>(content)
ct-tag-number = 1668546817..1668612095
; or use 0x63740101..0x6374FFFF
```

> Notes:
>
> 1. This syntax reuses the angle bracket syntax for generics; this reuse is innocuous because a generic parameter or argument only ever occurs after a rule name (`id`), while it occurs after the "`.`" (dot) character here. (Whether there is potential for human confusion can be debated; the above example deliberately uses generics as well.)
> 2. The updated ABNF grammar makes it a bit more explicit that the number given after the optional dot is the value of the argument: for tags and simple values, it is not giving the CBOR "additional information", as it is with other uses of # in CDDL. (Adding this observation to Section 2.2.3 of [RFC8610] is the subject of [Err6575]; it is correctly noted in Section 3.6 of [RFC8610].) In hindsight, maybe a different character than the dot should have been chosen for this special case; however, changing the grammar in the current document would have been too disruptive.

4. Security Considerations

The grammar fixes and updates in this document are not believed to create additional security considerations. The security considerations in Section 5 of [RFC8610] apply. Specifically, the potential for confusion is increased in an environment that uses a combination of CDDL tools, some of which have been updated and some of which have not, in particular based on Section 2.

Attackers may want to exploit such potential confusion by crafting CDDL models that are interpreted differently by different parts of a system. There will be a period of transition from the details that the grammar in [RFC8610] handled in a less well-defined way, to the updated grammar defined in the present document. This transition might offer one (but not the only) type of opportunity for the kind of attack that relies on differences between implementations. Implementations that make use of CDDL models operationally already need to ascertain the provenance (and thus authenticity and integrity) and applicability of models they employ. At the time of writing, it is expected that the models will generally be processed by a software developer, within a software development environment. Therefore, developers are advised to treat CDDL models with the same care as any other source code.

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[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>.

[STD94] Internet Standard 94, <https://www.rfc-editor.org/info/std94>.
At the time of writing, this STD comprises the following:

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

6.2. Informative References

[CDDL-MODULES] Bormann, C. and B. Moran, "CDDL Module Structure", Work in Progress, Internet-Draft, draft-ietf-cbor-cddl-modules-03, 1 September 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-cddl-modules-03>.

[EDN-LITERALS] Bormann, C., "CBOR Extended Diagnostic Notation (EDN)", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-literals-13, 3 November 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-literals-13>.

[Err6278] RFC Errata, Erratum ID 6278, RFC 8610, <https://www.rfc-editor.org/errata/eid6278>.

[Err6526] RFC Errata, Erratum ID 6526, RFC 8610, <https://www.rfc-editor.org/errata/eid6526>.

[Err6527] RFC Errata, Erratum ID 6527, RFC 8610, <https://www.rfc-editor.org/errata/eid6527>.

[Err6543] RFC Errata, Erratum ID 6543, RFC 8610, <https://www.rfc-editor.org/errata/eid6543>.

[Err6575] RFC Errata, Erratum ID 6575, RFC 8610, <https://www.rfc-editor.org/errata/eid6575>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/info/rfc7405>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/info/rfc9165>.

[RFC9277] Richardson, M. and C. Bormann, "On Stable Storage for Items in Concise Binary Object Representation (CBOR)", RFC 9277, DOI 10.17487/RFC9277, August 2022, <https://www.rfc-editor.org/info/rfc9277>.

[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>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <https://www.unicode.org/versions/latest/>.

Appendix A. Updated Collected ABNF for CDDL

This appendix is normative.

It provides the full ABNF from [RFC8610] as updated by the present document.

```
cddl = S *(rule S)
rule = typename [genericparm] S assignt S type
     / groupname [genericparm] S assigng S grpent

typename = id
groupname = id

assignt = "=" / "/="
assigng = "=" / "//="

genericparm = "<" S id S *("," S id S ) ">"
genericarg = "<" S type1 S *("," S type1 S ) ">"

type = type1 *(S "/" S type1)

type1 = type2 [S (rangeop / ctlop) S type2]
; space may be needed before the operator if type2 ends in a name

type2 = value
      / typename [genericarg]
      / "(" S type S ")"
      / "{" S group S "}"
      / "[" S group S "]"
      / "~" S typename [genericarg]
      / "&" S "(" S group S ")"
      / "&" S groupname [genericarg]
      / "#" "6" ["." head-number] "(" S type S ")"
      / "#" "7" ["." head-number]
      / "#" DIGIT ["." uint]                ; major/ai
      / "#"                                 ; any
head-number = uint / ("<" type ">")

rangeop = "..." / ".."

ctlop = "." id

group = grpchoice *(S "//" S grpchoice)

grpchoice = *(grpent optcom)

grpent = [occur S] [memberkey S] type
       / [occur S] groupname [genericarg]  ; preempted by above
       / [occur S] "(" S group S ")"

memberkey = type1 S ["^" S] "=>"
          / bareword S ":"
          / value S ":"

bareword = id

optcom = S ["," S]

occur = [uint] "*" [uint]
      / "+"
      / "?"

uint = DIGIT1 *DIGIT
```

```
     / "0x" 1*HEXDIG
     / "0b" 1*BINDIG
     / "0"

value = number
      / text
      / bytes

int = ["-"] uint

; This is a float if it has fraction or exponent; int otherwise
number = hexfloat / (int ["." fraction] ["e" exponent ])
hexfloat = ["-"] "0x" 1*HEXDIG ["." 1*HEXDIG] "p" exponent
fraction = 1*DIGIT
exponent = ["+"/"-"] 1*DIGIT

text = %x22 *SCHAR %x22
SCHAR = %x20-21 / %x23-5B / %x5D-7E / NONASCII / SESC

SESC = "\" ( %x22 / "/" / "\" /                 ; \" \/ \\
             %x62 / %x66 / %x6E / %x72 / %x74 / ; \b \f \n \r \t
             (%x75 hexchar) )                   ; \uXXXX

hexchar = "{" (1*"0" [ hexscalar ] / hexscalar) "}" /
          non-surrogate / (high-surrogate "\" %x75 low-surrogate)
non-surrogate = ((DIGIT / "A"/"B"/"C" / "E"/"F") 3HEXDIG) /
                ("D" %x30-37 2HEXDIG )
high-surrogate = "D" ("8"/"9"/"A"/"B") 2HEXDIG
low-surrogate = "D" ("C"/"D"/"E"/"F") 2HEXDIG
hexscalar = "10" 4HEXDIG / HEXDIG1 4HEXDIG
          / non-surrogate / 1*3HEXDIG

bytes = [bsqual] %x27 *BCHAR %x27
BCHAR = %x20-26 / %x28-5B / %x5D-7E / NONASCII / SESC / "\'" / CRLF
bsqual = "h" / "b64"

id = EALPHA *(*("-" / ".") (EALPHA / DIGIT))
ALPHA = %x41-5A / %x61-7A
EALPHA = ALPHA / "@" / "_" / "$"
DIGIT = %x30-39
DIGIT1 = %x31-39
HEXDIG = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"
HEXDIG1 = DIGIT1 / "A" / "B" / "C" / "D" / "E" / "F"
BINDIG = %x30-31

S = *WS
WS = SP / NL
SP = %x20
NL = COMMENT / CRLF
COMMENT = ";" *PCHAR CRLF
PCHAR = %x20-7E / NONASCII
NONASCII = %xA0-D7FF / %xE000-10FFFD
CRLF = %x0A / %x0D.0A
```

Figure 11: ABNF for CDDL as Updated

Appendix B. Details about Covering Erratum ID 6543

This appendix is informative.

[Err6543] notes that the ABNF used in [RFC8610] for the content of byte string literals lumps together byte strings notated as text with byte strings notated in base16 (hex) or base64 (but see also updated BCHAR rule in Figure 4):

```
; ABNF from RFC 8610:
bytes = [bsqual] %x27 *BCHAR %x27
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
```

Figure 12: Original ABNF from RFC 8610 for BCHAR

B.1. Change Proposed by Erratum ID 6543

Erratum ID 6543 proposes handling the two cases in separate ABNF rules (where, with an updated SESC, BCHAR obviously needs to be updated as above):

```
; Proposal from Erratum ID 6543:
bytes = %x27 *BCHAR %x27
      / bsqual %x27 *QCHAR %x27
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
QCHAR = DIGIT / ALPHA / "+" / "/" / "-" / "_" / "=" / WS
```

Figure 13: Proposal from Erratum ID 6543 to Split the Byte String Rules

This potentially causes a subtle change, which is hidden in the WS rule:

```
; ABNF from RFC 8610:
WS = SP / NL
SP = %x20
NL = COMMENT / CRLF
COMMENT = ";" *PCHAR CRLF
PCHAR = %x20-7E / %x80-10FFFD
CRLF = %x0A / %x0D.0A
```

Figure 14: ABNF Definition of WS from RFC 8610

This allows any non-C0 character in a comment, so this fragment becomes possible:

```
foo = h'
   43424F52 ; 'CBOR'
   0A       ; LF, but don't use CR!
'
```

The current text is not unambiguously saying whether the three apostrophes need to be escaped with a `\` or not, as in:

```
foo = h'
   43424F52 ; \'CBOR\'
   0A       ; LF, but don\'t use CR!
'
```

... which would be supported by the existing ABNF in [RFC8610].

B.2. No Further Change Needed after Updating String Literal Grammar

This document takes the simpler approach of leaving the processing of the content of the byte string literal to a semantic step after processing the syntax of the `bytes` and `BCHAR` rules, as updated by Figures 2 and 4 in Section 2.1 (updates prompted by the combination of [Err6527] and [Err6278]).

Therefore, the rules in Figure 14 (as updated by Figure 4) are applied to the result of this processing where `bsqual` is given as `h` or `b64`.

Note that this approach also works well with the use of byte strings in Section 3 of [RFC9165]. It does require some care when copying-and-pasting into CDDL models from ABNF that contains single quotes (which may also hide as apostrophes in comments); these need to be escaped or possibly replaced by `%x27`.

Finally, the approach taken lends support to extending `bsqual` in CDDL similar to the way this is done for CBOR diagnostic notation in [EDN-LITERALS]. (Note that, at the time of writing, the processing of string literals is quite similar for both CDDL and Extended Diagnostic Notation (EDN), except that CDDL has end-of-line comments that are "`;`" based and EDN has two comment syntaxes: one in-line "`/`" based and one end-of-line "`#`" based.)

Acknowledgments

Many thanks go to the submitters of the errata reports addressed in this document. In one of the ensuing discussions, Doug Ewell proposed defining an ABNF rule "NONASCII", of which we have included the essence. Special thanks to the reviewers Marco Tiloca, Christian Amsüss (Shepherd Review and further guidance), Orie Steele (AD Review and further guidance), and Éric Vyncke (detailed IESG review).

Author's Address

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