

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 29 June 2022

C. Bormann
Universität Bremen TZI
26 December 2021

A feature freezer for the Concise Data Definition Language (CDDL)
draft-bormann-cbor-cddl-freezer-09

Abstract

In defining the Concise Data Definition Language (CDDL), some features have turned up that would be nice to have. In the interest of completing this specification in a timely manner, the present document was started to collect nice-to-have features that did not make it into the first RFC for CDDL, [RFC 8610](#), or the specifications exercising its extension points, such as [RFC 9165](#).

It is now time to discuss thawing some of the concepts discussed here. A number of additional proposals have been added.

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/>.

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 will expire on 29 June 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components

Internet-Draft

CDDL feature freezer

December 2021

extracted from this document must include Revised BSD License text as described in Section 4.e of the [Trust Legal Provisions](#) and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.	Introduction	2
2.	Base language features	3
2.1.	Cuts	3
3.	Literal syntax	3
3.1.	Tag-oriented Literals	3
3.2.	Regular Expression Literals	4
3.3.	Clarifications	4
3.3.1.	Err6527	4
3.3.2.	Err6543	5
4.	Controls	6
4.1.	Control operator .pcre	6
4.2.	Endianness in .bits	6
4.3.	.bitfield control	7
5.	Co-occurrence Constraints	7
6.	Module superstructure	8
6.1.	Namespacing	8
6.2.	Cross-universe references	8
6.2.1.	IANA references	8
6.3.	Potential examples	9
6.3.1.	How name spaces might look like	9
6.3.2.	Explicitly interacting with namespaces	9
6.3.3.	Document references	10
6.3.4.	Add retroactive exporting to RFCs	10
6.3.5.	Operations	10
6.3.6.	To be discussed	11
7.	Alternative Representations	11
8.	IANA Considerations	11
9.	Security considerations	11
10.	References	11
10.1.	Normative References	12
10.2.	Informative References	12
	Acknowledgements	14
	Author's Address	14

[1.](#) Introduction

In defining the Concise Data Definition Language (CDDL), some

features have turned up that would be nice to have. In the interest of completing this specification in a timely manner, the present document was started to collect nice-to-have features that did not make it into the first RFC for CDDL [[RFC8610](#)], or the specifications exercising its extension points, such as [[RFC9165](#)].

It is now time to discuss thawing some of the concepts discussed here. A number of additional proposals have been added.

There is always a danger for a document like this to become a shopping list; the intention is to develop this document further based on the rapidly growing real-world experience with the first CDDL standard.

[2.](#) Base language features

[2.1.](#) Cuts

[Section 3.5.4 of \[RFC8610\]](#) alludes to a new language feature, `_cuts_`, and defines it in a fashion that is rather focused on a single application in the context of maps and generating better diagnostic information about them.

The present document is expected to grow a more complete definition of cuts, with the expectation that it will be upwards-compatible to the existing one in [[RFC8610](#)], before this possibly becomes a mainline language feature in a future version of CDDL.

[3.](#) Literal syntax

[3.1.](#) Tag-oriented Literals

Some CBOR tags often would be most natural to use in a CDDL spec with a literal syntax that is tailored to their semantics instead of their serialization in CBOR. There is currently no way to add such syntaxes, no defined extension point either.

Based on the CoRAL work [[I-D.ietf-core-coral](#)], the proposal "Application-Oriented Literals in CBOR Extended Diagnostic Notation" [[I-D.bormann-cbor-edn-literals](#)] defines application-oriented literals, e.g., of the form

dt'2019-07-21T19:53Z'

for datetime items. With additional considerations for unambiguous syntax, a similar literal form could be included in CDDL.

Bormann

Expires 29 June 2022

[Page 3]

Internet-Draft

CDDL feature freezer

December 2021

[3.2.](#) Regular Expression Literals

Regular expressions currently are notated as strings in CDDL, with all the string escaping rules applied once. It might be convenient to have a more conventional literal format for regular expressions, possibly also providing a place to add modifiers such as `/i`. This might also imply text `.regexp ...`, which with the proposal in [Section 4.1](#) then raises the question of how to indicate the regular expression flavor.

[3.3.](#) Clarifications

A number of errata reports have been made around some details of text string and byte string literal syntax: [\[Err6527\]](#) and [\[Err6543\]](#). These need to be addressed by re-examining the details of these literal syntaxes. Also, [\[Err6526\]](#) needs to be applied.

[3.3.1.](#) Err6527

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

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

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. Both can be solved by

updating the SESC production to:

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

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

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

The updated version explicit allows \', which is no longer allowed in the updated SESC:

```
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / "\"'\" / CRLF
```

[3.3.2.](#) Err6543

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 production above):

```
bytes = [bsqual] %x27 *BCHAR %x27  
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
```

Errata report 6543 proposes to handle the two cases in separate productions (where, with an updated SESC, BCHAR obviously needs to be updated as above):

```
bytes = %x27 *BCHAR %x27  
       / bsqual %x27 *QCHAR %x27  
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
```

QCHAR = DIGIT / ALPHA / "+" / "/" / "-" / "_" / "=" / WS

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

```
WS = SP / NL
SP = %x20
NL = COMMENT / CRLF
COMMENT = ";" *PCHAR CRLF
PCHAR = %x20-7E / %x80-10FFFF
CRLF = %x0A / %x0D.0A
```

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 [\[RFC8610\]](#).

4. Controls

Controls are the main extension point of the CDDL language. It is relatively painless to add controls to CDDL; this mechanism has been exercised in [\[RFC9090\]](#) for SDNV [\[RFC6256\]](#) and ASN.1 OID related byte strings, and in [\[RFC9165\]](#) for more generally applicable controls, including an interface to ABNF [\[RFC5234\]](#) [\[RFC7405\]](#). Several further candidates have been identified that aren't quite ready for adoption,

of which a few shall be listed here.

[4.1.](#) Control operator `.pcre`

There are many variants of regular expression languages. [Section 3.8.3 of \[RFC8610\]](#) defines the `.regex` control, which is based on XSD [\[XSD2\]](#) regular expressions. As discussed in that section, the most desirable form of regular expressions in many cases is the family called "Perl-Compatible Regular Expressions" ([\[PCRE\]](#)); however, no formally stable definition of PCRE is available at this time for normatively referencing it from an RFC.

The present document defines the control operator `.pcre`, which is similar to `.regex`, but uses PCRE2 regular expressions. More specifically, a `.pcre` control indicates that the text string given as a target needs to match the PCRE regular expression given as a value in the control type, where that regular expression is anchored on both sides. (If anchoring is not desired for a side, `.*` needs to be inserted there.)

Similarly, `.es2018re` could be defined for ECMAScript 2018 regular expressions with anchors added.

See also [\[I-D.draft-bormann-jsonpath-iregexp\]](#), which could be specifically called out via `.iregexp` (even though `.regex` as per [Section 3.8.3 of \[RFC8610\]](#) would also have the same semantics, except for a wider range of regexps).

[4.2.](#) Endianness in `.bits`

How useful would it be to have another variant of `.bits` that counts bits like in RFC box notation? (Or at least per-byte? 32-bit words don't always perfectly mesh with byte strings.)

[4.3.](#) `.bitfield` control

Provide a way to specify bitfields in byte strings and uints to a higher level of detail than is possible with `.bits`. Strawman:

Field = uint `.bitfield` Fieldbits

Fieldbits = [

```
    flag1: [1, bool],
    val: [4, Vals],
    flag2: [1, bool],
]
```

```
Vals = &(A: 0, B: 1, C: 2, D: 3)
```

Note that the group within the controlling array can have choices, enabling the whole power of a context-free grammar (but not much more).

[5.](#) Co-occurrence Constraints

While there are no co-occurrence constraints in CDDL, many actual use cases can be addressed by using the fact that a group is a grammar:

```
postal = {
  ( street: text,
    housenumber: text) //
  ( pobox: text .regexp "[0-9]+" )
}
```

However, constraints that are not just structural/tree-based but are predicates combining parts of the structure cannot be expressed:

```
session = {
  timeout: uint,
}

other-session = {
  timeout: uint .lt [somehow refer to session.timeout],
}
```

As a minimum, this requires the ability to reach over to other parts of the tree in a control. Compare JSON Pointer [[RFC6901](#)] and JSON Relative Pointer [[I-D.handrews-relative-json-pointer](#)]. Stefan Goessner's jsonpath is a JSON variant of XPath that has not been formally standardized yet [[jsonpath](#)].

be needed.

[6.](#) Module superstructure

CDDL rules could be packaged as modules and referenced from other modules. There could be some control of namespace pollution, as well as unambiguous referencing ("versioning").

This is probably best achieved by a pragma-like syntax which could be carried in CDDL comments, leaving each module to be valid CDDL (if missing some rule definitions to be imported).

[6.1.](#) Namespacing

A convention for mapping CDDL-internal names to external ones could be developed, possibly steered by some pragma-like constructs. External names would likely be URI-based, with some conventions as they are used in RDF or Curies. Internal names might look similar to XML QNames. Note that the identifier character set for CDDL deliberately includes \$ and @, which could be used in such a convention.

[6.2.](#) Cross-universe references

Often, a CDDL specification needs to import from specifications in a different language or platform.

[6.2.1.](#) IANA references

In many cases, CDDL specifications make use of values that are specified in IANA registries. The `.iana` control operator can be used to reference such a set of values.

The reference needs to be able to point to a draft, the registry of which has not been established yet, as well as to an established IANA registry.

An example of such a usage might be:

```
cose-algorithm = int .iana ["cose", "algorithms", "value"]
```

Unfortunately, the vocabulary employed in IANA registries has not been designed for machine references. In this case, the potential values would come from applying the XPath expression

```
//iana:registry[@id='algorithms']/iana:record/iana:value
```

to <https://www.iana.org/assignments/cose/cose.xml>, plus some filtering on the records returned that only leaves actual allocations. Additional functionality may be needed for filtering with respect to other columns of the registry record, e.g., <capabilities> in the case of this example.

[6.3.](#) Potential examples

This section shows some examples that illustrate potential syntaxes and semantics to be examined.

One of the potential objectives here is to keep documents that make use of this extension generally valid as CDDL 1.0 documents, albeit possibly with a need to add further CDDL 1.0 rules to obtain a complete specification.

[6.3.1.](#) How name spaces might look like

Implicit namespacing might be provided by using a document reference as a namespace tag:

```
RFC8610.int ↔ int  
RFC9090.oid ↔ oid
```

Note that this example establishes a namespace for the prelude ([RFC8610.int](#)); maybe it is worth to do that more explicitly.

[6.3.2.](#) Explicitly interacting with namespaces

New syntax for explicitly interacting with namespaces might be but into [RFC 8610](#) comments, with a specific prefix (and possibly starting left-aligned). Prefixes proposed include ;;< and ;#; the below will use ;# even though that probably could pose too many conflicts; it also might be too inconspicuous.

```
;;# export oid, roid, pen as RFC9090  
oid = #6.111(bstr)  
roid = #6.110(bstr)  
pen = #6.112(bstr)
```

Besides an implicit import such as

```
; unadorned, just import?  
a = [RFC9090.oid]
```

there also could be an explicit import syntax:

Internet-Draft

CDDL feature freezer

December 2021

```
;  
# import oid from RFC9090  
a = [oid]
```

Such an explicit syntax might also be able to provide additional parameters such as in the IANA examples above.

[6.3.3.](#) Document references

A convention for establishing RFC references might be easy to establish, but at least Internet-Draft references and IANA registry references should also supported. It is probably worth to add some indirection here, as names of Internet-Drafts might change (including by becoming RFCs).

[6.3.4.](#) Add retroactive exporting to RFCs

Existing RFCs with CDDL in them could presume an export ...all... as RFCnnnn (Possibly also per-section exports as in [RFC8610](#).D for the prelude?)

Namespace tags for those exports need to be reserved so they cannot be occupied by explicit exporting.

New specifications (including RFCs) can "include"/"import" from these namespaces, and maybe "export" their own rules in a more considered way.

[6.3.5.](#) Operations

★ "export":

1. prefix: add a namespace name to "local" rulenames:

```
`oid` → `RFC9090.oid`
```

2. make that namespace available to other specs

★ "import": include (prefixed) definitions from a source

1. use as is: [RFC9090](#).oid

2. unprefix: oid

Example: prelude processing -- include+unprefix from [Appendix D of RFC8610](#).

* "include": find files, turn into namespaces to import from

Bormann

Expires 29 June 2022

[Page 10]

Internet-Draft

CDDL feature freezer

December 2021

[6.3.6](#). To be discussed

How to find the document that exports a namespace (IANA? Use by other SDOs? Internal use in an org? How to transition between these states?)

Multiple documents exporting into one namespace (_Immutable_ [RFC9090](#) namespace vs. "OID"-namespace? Who manages _mutable_ namespaces?)

Updates, revisions, versions, semver:

;# insert OID ~> 2.2 ; twiddle-wakka: this version or higher

[7](#). Alternative Representations

For CDDL, alternative representations e.g. in JSON (and thus in YAML) could be defined, similar to the way YANG defines an XML-based serialization called YIN in [Section 11 of \[RFC6020\]](#). One proposal for such a syntax is provided by the cddltool [cddltool], which is reproduced below. This could be written up in more detail and agreed upon.

```
cddltool = ["cddl", +rule]
rule = ["=" / "/" = / "/" =, namep, type]
namep = ["name", id] / ["gen", id, +id]
id = text .regexp "[A-Za-z@_$_]([-.])*[A-Za-z0-9@_$_]*"
op = ".." / "..." /
    text .regexp "\\.[A-Za-z@_$_]([-.])*[A-Za-z0-9@_$_]*"
namea = ["name", id] / ["gen", id, +type]
type = value / namea / ["op", op, type, type] /
    ["map", group] / ["ary", group] / ["tcho", 2*type] /
    ["unwrap", namea] / ["enum", group / namea] /
```

```
["prim",?(0..7,?uint)]
group = ["mem", null/type, type] /
["rep", uint, uint/false, group] /
["seq", 2*group] / ["gcho", 2*group]
value = ["number"/"text"/"bytes", text]
```

[8.](#) IANA Considerations

This document makes no requests of IANA.

[9.](#) Security considerations

The security considerations of [[RFC8610](#)] apply.

[10.](#) References

Bormann	Expires 29 June 2022	[Page 11]
---------	----------------------	-----------

Internet-Draft	CDDL feature freezer	December 2021
----------------	----------------------	---------------

[10.1.](#) 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>>.
- [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>>.

[10.2.](#) Informative References

- [cddl-c] "CDDL conversion utilities", n.d., <<https://github.com/cabo/cddl-c>>.
- [Err6526] "Errata Report 6526", [RFC 8610](#), <<https://www.rfc-editor.org/errata/eid6526>>.
- [Err6527] "Errata Report 6527", [RFC 8610](#), <<https://www.rfc-editor.org/errata/eid6527>>.
- [Err6543] "Errata Report 6543", [RFC 8610](#),

<<https://www.rfc-editor.org/errata/eid6543>>.

[I-D.bormann-cbor-edn-literals]

Bormann, C., "Application-Oriented Literals in CBOR Extended Diagnostic Notation", Work in Progress, Internet-Draft, [draft-bormann-cbor-edn-literals-00](#), 6 October 2021, <<https://www.ietf.org/archive/id/draft-bormann-cbor-edn-literals-00.txt>>.

[I-D.[draft-bormann-jsonpath-iregexp](#)]

Bormann, C., "I-Regexp: An Interoperable Regexp Format", Work in Progress, Internet-Draft, [draft-bormann-jsonpath-iregexp-01](#), 13 November 2021, <<https://www.ietf.org/archive/id/draft-bormann-jsonpath-iregexp-01.txt>>.

[I-D.handrews-relative-json-pointer]

Luff, G. and H. Andrews, "Relative JSON Pointers", Work in Progress, Internet-Draft, [draft-handrews-relative-json-pointer-02](#), 18 September 2019, <<https://www.ietf.org/archive/id/draft-handrews-relative-json-pointer-02.txt>>.

Bormann

Expires 29 June 2022

[Page 12]

Internet-Draft

CDDL feature freezer

December 2021

[I-D.ietf-core-coral]

Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, [draft-ietf-core-coral-04](#), 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-coral-04.txt>>.

[jsonpath] "jsonpath online evaluator", n.d., <<https://jsonpath.com>>.

[PCRE]

"Perl-compatible Regular Expressions (revised API: PCRE2)", n.d., <<http://pcre.org/current/doc/html/>>.

[RFC5234]

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>>.

[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/info/rfc6020>>.

- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", [RFC 6256](#), DOI 10.17487/RFC6256, May 2011, <<https://www.rfc-editor.org/info/rfc6256>>.
- [RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", [RFC 6901](#), DOI 10.17487/RFC6901, April 2013, <<https://www.rfc-editor.org/info/rfc6901>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", [RFC 7405](#), DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/info/rfc7405>>.
- [RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", [RFC 9090](#), DOI 10.17487/RFC9090, July 2021, <<https://www.rfc-editor.org/info/rfc9090>>.
- [XSD2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, 28 October 2004, <<https://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.

Bormann

Expires 29 June 2022

[Page 13]

Internet-Draft

CDDL feature freezer

December 2021

Acknowledgements

Many people have asked for CDDL to be completed, soon. These are usually also the people who have brought up observations that led to the proposals discussed here. Sean Leonard has campaigned for a regex literal syntax.

Author's Address

Carsten Bormann
Universität Bremen TZI

Postfach 330440
D-28359 Bremen
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org