Network Working Group
Internet-Draft
Intended status: Informational
Expires: 26 August 2021

C. Bormann
Universität Bremen TZI

22 February 2021

A feature freezer for the Concise Data Definition Language (CDDL)
draft-bormann-cbor-cddl-freezer-05

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.

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

Table of Contents

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].

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 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.

The text form of CoRAL [I-D.ietf-core-coral] defines literals of the form

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

for datetime items. (Similar advances should then probably be made in diagnostic notation.)

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.

4. Controls

Controls are the main extension point of the CDDL language. It is relatively painless to add controls to CDDL. Several candidates have been identified that aren't quite ready for adoption, of which one 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 .regexp 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 .regexp, 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.

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 [jsonpath].

More generally, something akin to what Schematron is to Relax-NG may 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.

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 "cddlc" tool [cddlc]; this could be written up and agreed upon.

```
cddlj = ["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

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>.

10.2. Informative References

[cddlc] "CDDL conversion utilities", n.d., <https://github.com/cabo/cddlc>.

[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>.

[I-D.ietf-core-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-03, 9 March 2020, <https://www.ietf.org/archive/id/draft-ietf-core-coral-03.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/>.

[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>.

[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>.

[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>.

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 regexp 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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 23 October 2021

C. Bormann
Universität Bremen TZI

21 April 2021

A feature freezer for the Concise Data Definition Language (CDDL)
draft-bormann-cbor-cddl-freezer-07

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.

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

Table of Contents

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].

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 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.

The text form of CoRAL [I-D.ietf-core-coral] defines literals of the form

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

for datetime items. (Similar advances should then probably be made in diagnostic notation.)

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 / %x80-10FFFD
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. Several candidates have been identified that aren't quite ready for adoption, of which one 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 .regexp 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 .regexp, 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.

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 [jsonpath].

More generally, something akin to what Schematron is to Relax-NG may 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.

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 "cddlc" tool [cddlc]; this could be written up and agreed upon.

```
cddlj = ["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

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>.

10.2. Informative References

[cddlc] "CDDL conversion utilities", n.d., <https://github.com/cabo/cddlc>.

[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.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>.

[I-D.ietf-core-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-03, 9 March 2020, <https://www.ietf.org/archive/id/draft-ietf-core-coral-03.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/>.

[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>.

[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>.

[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>.

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 regexp 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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 16 August 2021

C. Bormann
Universität Bremen TZI
12 February 2021

Notable CBOR Tags
draft-bormann-cbor-notable-tags-03

Abstract

The Concise Binary Object Representation (CBOR, RFC 7049) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

In CBOR, one point of extensibility is the definition of CBOR tags. RFC 7049 and its revision 7049bis define a basic set of tags as well as a registry that can be used to contribute additional tag definitions [IANA.cbor-tags]. Since RFC 7049 was published, some 80 tag definitions have been added to that registry.

The present document provides a roadmap to a large subset of these tag definitions. Where applicable, it points to a IETF standards or standard development document that specifies the tag. Where no such document exists, the intention is to collect specification information from the sources of the registrations. After some more development, the present document is intended to be useful as a reference document for the IANA registrations of the CBOR tags the definitions of which have been collected.

Note to Readers

This is an individual submission to the CBOR working group of the IETF, https://datatracker.ietf.org/wg/cbor/about/ (https://datatracker.ietf.org/wg/cbor/about/). Discussion currently takes places on the github repository https://github.com/cabo/notable-tags (https://github.com/cabo/notable-tags). If the CBOR WG believes this is a useful document, discussion is likely to move to the CBOR WG mailing list and a github repository at the CBOR WG github organization, https://github.com/cbor-wg (https://github.com/cbor-wg).

The current version is true work in progress; some of the sections haven't been filled in yet, and in particular, permission has not been obtained from tag definition authors to copy over their text.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

(TO DO, expand on text from abstract here; move references here and neuter them in the abstract as per Section 4.3 of [RFC7322].)

The selection of the tags presented here is somewhat arbitrary; considerations such as how wide the scope and area of application of a tag definition is combine with an assessment how "ready to use" the tag definition is (i.e., is the tag specification in a state where it can be used).

This document can only be a snapshot of a subset of the current registrations. The most up to date set of registrations is always available in the registry at [IANA.cbor-tags].

1.1. Terminology

The definitions of [RFC8949] apply. The term "byte" is used in its now customary sense as a synonym for "octet". Where bit arithmetic is explained, this document uses the notation familiar from the programming language C (including C++14's 0bnnn binary literals), except that the operator "**" stands for exponentiation.

2. RFC 7049 (CBOR)

[RFC7049] defines a number of tags that are listed here for convenience only.

Tag number	Tag content	Short Description	Section of RFC 7049
0	UTF-8 string	Standard date/time string	2.4.1
1	multiple	Epoch-based date/time	2.4.1
2	byte string	Positive bignum	2.4.2
3	byte string	Negative bignum	2.4.2
4	array	Decimal fraction	2.4.3
5	array	Bigfloat	2.4.3
21	multiple	Expected conversion to base64url encoding	2.4.4.2
22	multiple	Expected conversion to base64 encoding	2.4.4.2
23	multiple	Expected conversion to base16 encoding	2.4.4.2
24	byte string	Encoded CBOR data item	2.4.4.1
32	UTF-8 string	URI	2.4.4.3
33	UTF-8 string	base64url	2.4.4.3
34	UTF-8 string	base64	2.4.4.3
35	UTF-8 string	Regular expression	2.4.4.3
36	UTF-8 string	MIME message	2.4.4.3
55799	multiple	Self-describe CBOR	2.4.5

Table 1: Tag numbers defined in RFC 7049

2.1. Tags Related to Those Defined in RFC 7049

Separately registered tags that are directly related to the tags predefined in RFC 7049 include:

* Tag 63, registered by this document, is a parallel to tag 24, with the single difference that its byte string tag content carries a CBOR Sequence [RFC8742] instead of a single CBOR data items.

* Tag 257, registered by Peter Occil with a specification in http://peteroupc.github.io/CBOR/binarymime.html (http://peteroupc.github.io/CBOR/binarymime.html), is a parallel to tag 36, except that the tag content is a byte string, which therefore can also carry binary MIME messages as per [RFC2045].

3. Security

A number of CBOR tags are defined in security specifications that make use of CBOR.

3.1. RFC 8152 (COSE)

[RFC8152] defines CBOR Object Signing and Encryption (COSE). A revision is in process that splits this specification into the data structure definitions [I-D.ietf-cose-rfc8152bis-struct], which will define another tag for COSE standalone counter signature, and the algorithms employed [I-D.ietf-cose-rfc8152bis-algs].

Tag number	Tag content	Short Description
16	COSE_Encrypt0	COSE Single Recipient Encrypted Data Object
17	COSE_Mac0	COSE Mac w/o Recipients Object
18	COSE_Sign1	COSE Single Signer Data Object
96	COSE_Encrypt	COSE Encrypted Data Object
97	COSE_Mac	COSE MACed Data Object
98	COSE_Sign	COSE Signed Data Object

Table 2: Tag numbers defined in RFC 8152, COSE

3.2. RFC 8392 (CWT)

[RFC8392] defines the CBOR Web Token (CWT), making use of COSE to define a CBOR variant of the JOSE Web Token (JWT), [RFC7519], a standardized security token that has found use in the area of web applications, but is not technically limited to those.

Tag number	Tag content	Short Description
61	CBOR Web Token (CWT)	CBOR Web Token (CWT)

Table 3: Tag number defined for RFC 8392 CBOR Web Token (CWT)

4. CBOR-based Representation Formats

Representation formats can be built on top of CBOR.

4.1. YANG-CBOR

YANG [RFC7950] is a data modeling language originally designed in the context of the Network Configuration Protocol (NETCONF) [RFC6241], now widely used for modeling management and configuration information. [RFC7950] defines an XML-based representation format, and [RFC7951] defines a JSON-based [RFC8259] representation format for YANG.

YANG-CBOR [I-D.ietf-core-yang-cbor] is a representation format for YANG data in CBOR.

Tag number	Tag content	Short Description	Section of YANG-CBOR
43	byte string	YANG bits datatype	6.7
44	unsigned integer	YANG enumeration datatype	6.6
45	unsigned integer or text string	YANG identityref datatype	6.10
46	unsigned integer or text string or array	YANG instance-identifier datatype	6.13
47	unsigned integer	YANG Schema Item iDentifier (sid)	3.2

Table 4: Tag number defined for YANG-CBOR

5. Protocols

Protocols may want to allocate CBOR tag numbers to identify specific protocol elements.

5.1. DOTS

DDoS Open Threat Signaling (DOTS) defines tag number 271 for the DOTS signal channel object in [RFC8782].

5.2. RAINS

As an example for how experimental protocols can make use of CBOR tag definitions, the RAINS (Another Internet Naming Service) Protocol Specification defines tag number 15309736 for a RAINS Message [I-D.trammell-rains-protocol].

6. Datatypes

6.1. Advanced arithmetic

A number of tags have been registered for arithmetic representations beyond those built into CBOR and defined by tags in [RFC7049]. These are all documented under "http://peteroupc.github.io/CBOR/"; the last pathname component is given in Table 5.

(TO DO: Obtain permission to copy the definitions here.)

Tag number	Tag content	Short Description	Reference
30	array	Rational number	rational.html
264	array	Decimal fraction with arbitrary exponent	bigfrac.html
265	array	Bigfloat with arbitrary exponent	bigfrac.html
268	array	Extended decimal fraction	extended.html
269	array	Extended bigfloat	extended.html
270	array	Extended rational number	extended.html

Table 5: Tags for advanced arithmetic

6.2. Variants of undefined

"https://github.com/svaarala/cbor-specs/blob/master/cbor-absent-tag.rst" defines tag 31 to be applied to the CBOR value Undefined (0xf7), slightly modifying its semantics to stand for an absent value in a CBOR Array.

(TO DO: Obtain permission to copy the definitions here.)

6.3. Typed and Homogeneous Arrays

[RFC8746] defines tags for various kinds of arrays. A summary is reproduced in Table 6.

Tag	Data Item	Semantics
64	byte string	uint8 Typed Array
65	byte string	uint16, big endian, Typed Array
66	byte string	uint32, big endian, Typed Array
67	byte string	uint64, big endian, Typed Array
68	byte string	uint8 Typed Array, clamped arithmetic
69	byte string	uint16, little endian, Typed Array
70	byte string	uint32, little endian, Typed Array
71	byte string	uint64, little endian, Typed Array
72	byte string	sint8 Typed Array
73	byte string	sint16, big endian, Typed Array
74	byte string	sint32, big endian, Typed Array
75	byte string	sint64, big endian, Typed Array
76	byte string	(reserved)
77	byte string	sint16, little endian, Typed Array
78	byte string	sint32, little endian, Typed Array
79	byte string	sint64, little endian, Typed Array
80	byte string	IEEE 754 binary16, big endian, Typed Array
81	byte string	IEEE 754 binary32, big endian, Typed Array
82	byte string	IEEE 754 binary64, big endian, Typed Array
83	byte string	IEEE 754 binary128, big endian, Typed Array

84	byte string	IEEE 754 binary16, little endian, Typed Array
85	byte string	IEEE 754 binary32, little endian, Typed Array
86	byte string	IEEE 754 binary64, little endian, Typed Array
87	byte string	IEEE 754 binary128, little endian, Typed Array
40	array of two arrays*	Multi-dimensional Array, row-major order
1040	array of two arrays*	Multi-dimensional Array, column-major order
41	array	Homogeneous Array

Table 6: Tag numbers defined for Arrays

7. Domain-Specific

(TO DO: Obtain permission to copy the definitions here; create proper table.)

37	byte string	Binary UUID ([RFC4122] section 4.1.2)	[https://github.com/lucas-clemente/cbor-specs/blob/master/uuid.md][Lucas_Clemente]
38	array	Language-tagged string	[http://peteroupc.github.io/CBOR/langtags.html][Peter_Occil]
257	byte string	Binary MIME message	[http://peteroupc.github.io/CBOR/binarymime.html][Peter_Occil]
260	byte string	Network Address (IPv4 or IPv6 or MAC Address)	[http://www.employees.org/~ravir/cbor-network.txt][Ravi_Raju]
261	map (IPAddress + Mask Length)	Network Address Prefix (IPv4 or IPv6 Address + Mask Length)	[https://github.com/toravir/CBOR-Tag-Specs/blob/master/networkPrefix.md][Ravi_Raju]
263	byte string	Hexadecimal string	[https://github.com/toravir/CBOR-Tag-Specs/blob/master/hexString.md][Ravi_Raju]
266	text string	Internationalized resource identifier (IRI)	[https://peteroupc.github.io/CBOR/iri.html][Peter_Occil]
267	text string	Internationalized resource identifier reference (IRI reference)	[https://peteroupc.github.io/CBOR/iri.html][Peter_Occil]

7.1. Extended Time Formats

Additional tag definitions have been provided for date and time values.

Tag	Data Item	Semantics	Reference
100	integer	date in number of days since epoch	[RFC8943]
1004	text string	RFC 3339 full-date string	[RFC8943]
1001	map	extended time	[I-D.bormann-cbor-time-tag]
1002	map	duration	[I-D.bormann-cbor-time-tag]
1003	map	period	[I-D.bormann-cbor-time-tag]

Table 7: Tag numbers for date and time

Note that tags 100 and 1004 are for calendar dates that are not anchored to a specific time zone; they are meant to specify calendar dates as perceived by humans, e.g. for use in personal identification documents. Converting such a calendar date into a specific point in time needs the addition of a time-of-day (for which a CBOR tag is outstanding) and timezone information (also outstanding). Alternatively, a calendar date plus timezone information can be converted into a time period (range of time values given by the starting and the ending time); note that these time periods are not always exactly 24 h (86400 s) long.

[RFC8943] does not suggest CDDL [RFC8610] type names for the two tags. We suggest copying the definitions in Figure 1 into application-specific CDDL as needed.

```
caldate = #6.100(int) ; calendar date as a number of days from 1970-01-01
tcaldate = #6.1004(tstr) ; calendar date as an RFC 3339 full-date string
```

Figure 1: CDDL for calendar date tags (RFC8943)

Tag 1001 extends tag 1 by additional information (such as picosecond resolution) and allows the use of Decimal and Bigfloat numbers for the time.

8. Platform-oriented

8.1. Perl

(These are actually not as Perl-specific as the title of this section suggests. See also the penultimate paragraph of Section 3.4 of [RFC8949].)

These are all documented under "http://cbor.schmorp.de/"; the last pathname component is given in Table 8.

(TO DO: Obtain permission to copy the definitions here.)

Tag	Data Item	Semantics	Reference
256	multiple	mark value as having string references	stringref
25	unsigned integer	reference the nth previously seen string	stringref
26	array	Serialised Perl object with classname and constructor arguments	perl-object
27	array	Serialised language-independent object with type name and constructor arguments	generic-object
28	multiple	mark value as (potentially) shared	value-sharing
29	unsigned integer	reference nth marked value	value-sharing
22098	multiple	hint that indicates an additional level of indirection	indirection

Table 8: Tag numbers that aid the Perl platform

8.2. JSON

(TO DO: Obtain permission to copy the definitions here.)

Tag number 262 has been registered to identify byte strings that carry embedded JSON text ("https://github.com/toravir/CBOR-Tag-Specs/blob/master/embeddedJSON.md").

Tag number 275 can be used to identify maps that contain keys that are all of type Text String, as they would occur in JSON ("https://github.com/ecorm/cbor-tag-text-key-map").

8.3. Weird text encodings

(TO DO: Obtain permission to copy the definitions here.)

Some variants of UTF-8 are in use in specific areas of application. Tags have been registered to be able to carry around strings in these variants in case they are not also valid UTF-8 and can therefore not be represented as a CBOR text string ("https://github.com/svaarala/cbor-specs/blob/master/cbor-nonutf8-string-tags.rst").

Tag Number	Data Item	Semantics
272	byte string	Non-UTF-8 CESU-8 string
273	byte string	Non-UTF-8 WTF-8 string
274	byte string	Non-UTF-8 MUTF-8 string

Table 9: Tag numbers for UTF-8 variants

9. Application-specific

(TO DO: Obtain permission to copy the definitions here; create proper table.)

39	multiple	Identifier	[https://github.com/lucas-clemente/cbor-specs/blob/master/id.md][Lucas_Clemente]
42	byte string	IPLD content identifier	[https://github.com/ipld/cid-cbor/][Volker_Mische]
103	array	Geographic Coordinates	[https://github.com/allthingstalk/cbor/blob/master/CBOR-Tag103-Geographic-Coordinates.md][Danilo_Vidovic]
104	multiple	Geographic Coordinate Reference System WKT or EPSG number	[draft-clarke-cbor-crs]
120	multiple	Internet of Things Data Point	[https://github.com/allthingstalk/cbor/blob/master/CBOR-Tag120-Internet-of-Things-Data-Points.md][Danilo_Vidovic]
258	array	Mathematical finite set	[https://github.com/input-output-hk/cbor-sets-spec/blob/master/CBOR_SETS.md][Alfredo_Di_Napoli]
259	map	Map datatype with key-value operations (e.g. '.get()/.set()/.delete()')	[https://github.com/shanewholloway/js-cbor-codec/blob/master/docs/CBOR-259-spec--explicit-maps.md][Shane_Holloway]

10. Implementation aids

10.1. Invalid Tag

The present document registers tag numbers 65535, 4294967295, and 18446744073709551615 (16-bit 0xffff, 32-bit 0xffffffff, and 64-bit 0xffffffffffffffff) as Invalid Tags, tags that are always invalid, independent of the tag content provided. The purpose of these tag number registrations is to enable the tag numbers to be reserved for internal use by implementations to note the absence of a tag on a data item where a tag could also be expected with that data item as tag content.

The Invalid Tags are not intended to ever occur in interchanged CBOR data items. Generic CBOR decoder implementations are encouraged to raise an error if an Invalid Tag occurs in a CBOR data item even if there is no validity checking implemented otherwise.

11. IANA Considerations

In the registry [IANA.cbor-tags], IANA has allocated the first to third tag in Table 10 from the FCFS space, with the present document as the specification reference. IANA has allocated the fourth tag from the Specification Required space, with the present document as the specification reference.

Tag	Data Item	Semantics	Reference
65535	(none valid)	always invalid	draft-bormann-cbor-notable-tags, Section 10.1
4294967295	(none valid)	always invalid	draft-bormann-cbor-notable-tags, Section 10.1
18446744073709551615	(none valid)	always invalid	draft-bormann-cbor-notable-tags, Section 10.1
63	byte string	Encoded CBOR Sequence [RFC8742]	draft-bormann-cbor-notable-tags, Section 2.1

Table 10: Values for Tags

12. Security Considerations

The security considerations of [RFC8949] apply; the tags discussed here may also have specific security considerations that are mentioned in their specific sections above.

13. References

13.1. Normative References

[I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-15, 25 January 2021, <http://www.ietf.org/internet-drafts/draft-ietf-core-yang-cbor-15.txt>.

[IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[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>.

[RFC8746] Bormann, C., Ed., "Concise Binary Object Representation (CBOR) Tags for Typed Arrays", RFC 8746, DOI 10.17487/RFC8746, February 2020, <https://www.rfc-editor.org/info/rfc8746>.

[RFC8782] Reddy.K, T., Ed., Boucadair, M., Ed., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 8782, DOI 10.17487/RFC8782, May 2020, <https://www.rfc-editor.org/info/rfc8782>.

[RFC8949] 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>.

13.2. Informative References

[I-D.bormann-cbor-time-tag]
Bormann, C., Gamari, B., and H. Birkholz, "Concise Binary Object Representation (CBOR) Tags for Time, Duration, and Period", Work in Progress, Internet-Draft, draft-bormann-cbor-time-tag-03, 9 March 2020, <http://www.ietf.org/internet-drafts/draft-bormann-cbor-time-tag-03.txt>.

[I-D.ietf-cose-rfc8152bis-algs]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-algs-12, 24 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-cose-rfc8152bis-algs-12.txt>.

[I-D.ietf-cose-rfc8152bis-struct]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-14, 24 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-cose-rfc8152bis-struct-14.txt>.

[I-D.trammell-rains-protocol]
Trammell, B. and C. Fehlmann, "RAINS (Another Internet Naming Service) Protocol Specification", Work in Progress, Internet-Draft, draft-trammell-rains-protocol-05, 29 January 2019, <http://www.ietf.org/internet-drafts/draft-trammell-rains-protocol-05.txt>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <https://www.rfc-editor.org/info/rfc7322>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC8943] Jones, M., Nadalin, A., and J. Richter, "Concise Binary Object Representation (CBOR) Tags for Date", RFC 8943, DOI 10.17487/RFC8943, November 2020, <https://www.rfc-editor.org/info/rfc8943>.

Acknowledgements

Contributors

Peter Occil

Email: poccil14 at gmail dot com

Many More
To do

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 26 August 2021

C. Bormann
Universität Bremen TZI
B. Gamari
Well-Typed
H. Birkholz
Fraunhofer SIT

22 February 2021

Concise Binary Object Representation (CBOR) Tags for Time, Duration, and Period

draft-bormann-cbor-time-tag-04

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

In CBOR, one point of extensibility is the definition of CBOR tags. RFC 8949 defines two tags for time: CBOR tag 0 (RFC3339 time as a string) and tag 1 (Posix time as int or float). Since then, additional requirements have become known. The present document defines a CBOR tag for time that allows a more elaborate representation of time, as well as related CBOR tags for duration and time period. It is intended as the reference document for the IANA registration of the CBOR tags defined.

Note to Readers

Version -00 of the present draft opened up the possibilities provided by extended representations of time in CBOR. Version -01 consolidated this draft to non-speculative content, the normative parts of which are believed will stay unchanged during further development of the draft. This version is provided to aid the registration of the CBOR tag immediately needed. Versions -02 and -03 made use of the IANA allocations registered and made other editorial updates. Further versions will re-introduce some of the material from -00, but in a more concrete form.

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

The Concise Binary Object Representation (CBOR, [RFC8949]) provides for the interchange of structured data without a requirement for a pre-agreed schema. RFC 8949 defines a basic set of data types, as well as a tagging mechanism that enables extending the set of data types supported via an IANA registry.

(TBD: Expand on text from abstract here.)

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The term "byte" is used in its now customary sense as a synonym for "octet". Where bit arithmetic is explained, this document uses the notation familiar from the programming language C (including C++14's 0bnnn binary literals), except that the operator "**" stands for exponentiation.

2. Objectives

For the time tag, the present specification addresses the following objectives that go beyond the original tags 0 and 1:

* Additional resolution for epoch-based time (as in tag 1). CBOR tag 1 only provides for integer and up to binary64 floating point representation of times, limiting resolution to approximately microseconds at the time of writing (and progressively becoming worse over time).

* Indication of time scale. Tags 0 and 1 are for UTC; however, some interchanges are better performed on TAI. Other time scales may be registered once they become relevant (e.g., one of the proposed successors to UTC that might no longer use leap seconds, or a scale based on smeared leap seconds).

Not currently addressed, but possibly covered by the definition of additional map keys for the map inside the tag:

* Direct representation of natural platform time formats. Some platforms use epoch-based time formats that require some computation to convert them into the representations allowed by tag 1; these computations can also lose precision and cause ambiguities. (TBD: The present specification does not take a position on whether tag 1 can be "fixed" to include, e.g., Decimal or BigFloat representations. It does define how to use these with the extended time format.)

* Additional indication of intents about the interpretation of the time given, in particular for future times. Intents might include information about time zones, daylight savings times, etc.

Additional tags are defined for durations and periods.

3. Time Format

An extended time is indicated by CBOR tag 1001, which tags a map data item (CBOR major type 5). The map may contain integer (major types 0 and 1) or text string (major type 3) keys, with the value type determined by each specific key. Implementations MUST ignore key/value types they do not understand for negative integer and text string values of the key. Not understanding key/value for unsigned keys is an error.

The map must contain exactly one unsigned integer key, which specifies the "base time", and may also contain one or more negative integer or text-string keys, which may encode supplementary information such as:

* a higher precision time offset to be added to the base time,

* a reference time scale and epoch different from the default UTC and 1970-01-01

* information about clock quality parameters, such as source, accuracy, and uncertainty

Future keys may add:

* intent information such as timezone and daylight savings time, and/or possibly positioning coordinates, to express information that would indicate a local time.

While this document does not define supplementary text keys, a number of unsigned and negative-integer keys are defined below.

3.1. Key 1

Key 1 indicates a value that is exactly like the data item that would be tagged by CBOR tag 1 (Posix time [TIME_T] as int or float). The time value indicated by the value under this key can be further modified by other keys.

3.2. Keys 4 and 5

Keys 4 and 5 are like key 1, except that the data item is an array as defined for CBOR tag 4 or 5, respectively. This can be used to include a Decimal or Bigfloat epoch-based float [TIME_T] in an extended time.

3.3. Keys -3, -6, -9, -12, -15, -18

The keys -3, -6, -9, -12, -15 and -18 indicate additional decimal fractions by giving an unsigned integer (major type 0) and scaling this with the scale factor 1e-3, 1e-6, 1e-9, 1e-12, 1e-15, and 1e-18, respectively (see Table 1). More than one of these keys MUST NOT be present in one extended time data item. These additional fractions are added to a base time in seconds [SI-SECOND] indicated by a Key 1, which then MUST also be present and MUST have an integer value.

Key	meaning	example usage
-3	milliseconds	Java time
-6	microseconds	(old) UNIX time
-9	nanoseconds	(new) UNIX time
-12	picoseconds	Haskell time
-15	femtoseconds	(future)
-18	attoseconds	(future)

Table 1: Key for decimally scaled Fractions

3.4. Key -1: Time Scale

Key -1 is used to indicate a time scale. The value 0 indicates UTC, with the POSIX epoch [TIME_T]; the value 1 indicates TAI, with the PTP (Precision Time Protocol) epoch [IEEE1588-2008].

If key -1 is not present, time scale value 0 is implied. Additional values can be registered in the (TBD define name for time scale registry); values MUST be integers or text strings.

(Note that there should be no time scales "GPS" or "NTP" -- instead, the time should be converted to TAI or UTC using a single addition or subtraction.)

```
t     =  t     -  2208988800
 utc      ntp

 t     =  t     +  315964819
  tai      gps
```

Figure 1: Converting Common Offset Time Scales

3.5. Clock Quality

A number of keys are defined to indicate the quality of clock that was used to determine the point in time.

The first three are analogous to "clock-quality-grouping" in [RFC8575], which is in turn based on the definitions in [IEEE1588-2008]; two more are specific to this document.

```
ClockQuality-group = (
  ? ClockClass => uint .size 1 ; PTP/RFC8575
  ? ClockAccuracy => uint .size 1 ; PTP/RFC8575
  ? OffsetScaledLogVariance => uint .size 2 ; PTP/RFC8575
  ? Uncertainty => ~time/~duration
  ? Guarantee => ~time/~duration
)
ClockClass = -2
ClockAccuracy = -4
OffsetScaledLogVariance = -5
Uncertainty = -7
Guarantee = -8
```

3.5.1. ClockClass (Key -2)

Key -2 (ClockClass) can be used to indicate the clock class as per Table 5 of [IEEE1588-2008]. It is defined as a one-byte integer as that is the ranged defined there.

3.5.2. ClockAccuracy (Key -4)

Key -4 (ClockAccuracy) can be used to indicate the clock accuracy as per Table 6 of [IEEE1588-2008]. It is defined as a one-byte integer as that is the ranged defined there. The range between 32 and 47 is a slightly distorted logarithmic scale from 25 ns to 1 s (see Figure 2); the number 254 is the value to be used if an unknown accuracy needs to be expressed.

$$\text{enum} \quad \text{approx } 48 \; + \; \left\lfloor 2 \cdot \log_{10} \left\{ \frac{\text{acc}}{\text{mathrm\{s\}}} \right\} - \text{epsilon} \right\rfloor_{\text{acc}}$$

Figure 2: Approximate conversion from accuracy to accuracy enumeration value

3.5.3. OffsetScaledLogVariance (Key -5)

Key -5 (OffsetScaledLogVariance) can be used to represent the variance exhibited by the clock when it has lost its synchronization with an external reference clock. The details for the computation of this characteristic are defined in Section 7.6.3 of [IEEE1588-2008].

3.5.4. Uncertainty (Key -7)

Key -7 (Uncertainty) can be used to represent a known measurement uncertainty for the clock, as a numeric value in seconds or as a duration (Section 4).

For this document, uncertainty is defined as in Section 2.2.3 of [GUM]: "parameter, associated with the result of a measurement, that characterizes the dispersion of the values that could reasonably be attributed to the measurand". More specifically, the value for this key represents the extended uncertainty for k = 2, in seconds.

3.5.5. Guarantee (Key -8)

Key -8 (Guarantee) can be used to represent a stated guarantee for the accuracy of the point in time, as a numeric value in seconds or as a duration (Section 4) representing the maximum allowed deviation from the true value.

While such a guarantee is unattainable in theory, existing standards such as [RFC3161] stipulate the representation of such guarantees, and therefore this format provides a way to represent them as well; the time value given is nominally guaranteed to not deviate from the actual time by more than the value of the guarantee, in seconds.

4. Duration Format

A duration is the length of an interval of time. Durations in this format are given in SI seconds, possibly adjusted for conventional corrections of the time scale given (e.g., leap seconds).

Except for using Tag 1002 instead of 1001, durations are structurally identical to time values. Semantically, they do not measure the time elapsed from a given epoch, but from the start to the end of (an otherwise unspecified) interval of time.

In combination with an epoch identified in the context, a duration can also be used to express an absolute time.

> (TBD: Clearly, ISO8601 durations are rather different; we do not want to use these.)

5. Period Format

A period is a specific interval of time, specified as either two times giving the start and the end of that interval, or as one of these two plus a duration.

They are given as an array of unwrapped time and duration elements, tagged with Tag 1003:

```
Period = #6.1003([
  start: ~Time / null
  end: ~Time / null
  ? duration: ~Duration / null
])
```

If the third array element is not given, the duration element is null. Exactly two out of the three elements must be non-null, this can be clumsily expressed in CDDL as:

```
Period = #6.1003([
  (start: ~Time,
   ((end: ~Time,
     ? duration: null) //
    (end: null,
     duration: ~Duration))) //
  (start: null,
   end: ~Time,
   duration: ~Duration)
])
```

> (Issue: should start/end be given the two-element treatment, or start/duration?)

6. CDDL typenames

For the use with the CBOR Data Definition Language, CDDL [RFC8610], the type names defined in Figure 3 are recommended:

```
etime = #6.1001({* (int/tstr) => any})
duration = #6.1002({* (int/tstr) => any})
period = #6.1003([~etime/null, ~etime/null, ~duration/null])
```

Figure 3: Recommended type names for CDDL

7. IANA Considerations

In the registry [IANA.cbor-tags], IANA has allocated the tags in Table 2 from the FCFS space, with the present document as the specification reference.

Tag	Data Item	Semantics
1001	map	[RFCthis] extended time
1002	map	[RFCthis] duration
1003	array	[RFCthis] period

Table 2: Values for Tags

IANA is requested to change the "Data Item" column for Tag 1003 from "map" to "array".

> (TBD: Add registry for time scales. Add registry for map keys and allocation policies for additional keys.)

8. Security Considerations

The security considerations of RFC 8949 apply; the tags introduced here are not expected to raise security considerations beyond those.

Time, of course, has significant security considerations; these include the exploitation of ambiguities where time is security relevant (e.g., for freshness or in a validity span) or the disclosure of characteristics of the emitting system (e.g., time zone, or clock resolution and wall clock offset).

9. References

9.1. Normative References

[GUM] Joint Committee for Guides in Metrology, "Evaluation of measurement data Guide to the expression of uncertainty in measurement", JCGM 100:2008, September 2008, <https://www.bipm.org/en/publications/guides/gum.html>.

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags>.

[IEEE1588-2008] IEEE, "1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", July 2008, <http://standards.ieee.org/findstds/standard/1588-2008.html>.

[RFC2119] 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>.

[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>.

[RFC8949] 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>.

[SI-SECOND] International Organization for Standardization (ISO), "Quantities and units Part 3: Space and time", ISO 80000-3, 1 March 2006.

[TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1-2008, 2016 Edition, 2016, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16>.

9.2. Informative References

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

[RFC8575] Jiang, Y., Ed., Liu, X., Xu, J., and R. Cummings, Ed., "YANG Data Model for the Precision Time Protocol (PTP)", RFC 8575, DOI 10.17487/RFC8575, May 2019, <https://www.rfc-editor.org/info/rfc8575>.

Acknowledgements

Authors' Addresses

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

Ben Gamari
Well-Typed
117 Middle Rd.
Portsmouth, NH 03801
United States

Email: ben@well-typed.com

Henk Birkholz
Fraunhofer Institute for Secure Information Technology
Rheinstrasse 75
64295 Darmstadt
Germany

Email: henk.birkholz@sit.fraunhofer.de

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 26 August 2021

C. Bormann
Universität Bremen TZI

22 February 2021

Additional Control Operators for CDDL
draft-ietf-cbor-cddl-control-02

Abstract

The Concise Data Definition Language (CDDL), standardized in RFC 8610, provides "control operators" as its main language extension point.

The present document defines a number of control operators that did not make it into RFC 8610: ".cat"/".plus" for the construction of constants, ".abnf"/".abnfb" for including ABNF (RFC 5234/RFC 7405) in CDDL specifications, and ".feature" for indicating the use of a non-basic feature in an instance.

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 26 August 2021.

Table of Contents

1. Introduction

The Concise Data Definition Language (CDDL), standardized in RFC 8610, provides "control operators" as its main language extension point.

The present document defines a number of control operators that did not make it into RFC 8610:

Name	Purpose
.cat	String Concatenation
.plus	Numeric addition
.abnf	ABNF in CDDL (text strings)
.abnfb	ABNF in CDDL (byte strings)
.feature	Detecting feature use in extension points

Table 1: New control operators in this document

1.1. Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses terminology from [RFC8610]. In particular, with respect to control operators, "target" refers to the left hand side operand, and "controller" to the right hand side operand.

2. Computed Literals

CDDL as defined in [RFC8610] does not have any mechanisms to compute literals. As an 80 % solution, this specification adds two control operators: ".cat" for string concatenation, and ".plus" for numeric addition.

2.1. String Concatenation

It is often useful to be able to compose string literals out of component literals defined in different places in the specification.

The ".cat" control identifies a string that is built from a concatenation of the target and the controller. As targets and controllers are types, the resulting type is formally the cross-product of the two types, although not all tools may be able to work with non-unique targets or controllers.

Target and controller MUST be strings. The result of the operation has the type of the target. The concatenation is performed on the bytes in both strings. If the target is a text string, the result of that concatenation MUST be valid UTF-8.

```
a = "foo" .cat '
  bar
  baz
'
; on a system where the newline is \n, is the same string as:
b = "foo\n  bar\n  baz\n"
```

Figure 1: Example: concatenation of text and byte string

The example in Figure 1 builds a text string named "a" out of concatenating the target text string ""foo"" and the controller byte string entered in a text form byte string literal. (This particular idiom is useful when the text string contains newlines, which, as

shown in the example for "b", may be harder to read when entered in the format that the pure CDDL text string notation inherits from JSON.)

2.2. Numeric Addition

In many cases in a specification, numbers are needed relative to a base number. The ".plus" control identifies a number that is constructed by adding the numeric values of the target and of the controller.

Target and controller MUST be numeric. If the target is a floating point number and the controller an integer number, or vice versa, the sum is converted into the type of the target; converting from a floating point number to an integer selects its floor (the largest integer less than or equal to the floating point number).

```
interval<BASE> = (
  BASE => int              ; lower bound
  (BASE .plus 1) => int    ; upper bound
  ? (BASE .plus 2) => int ; tolerance
)

X = 0
Y = 3
rect = {
  interval<X>
  interval<Y>
}
```

Figure 2: Example: addition to a base value

The example in Figure 2 contains the generic definition of a group "interval" that gives a lower and an upper bound and optionally a tolerance. "rect" combines two of these groups into a map, one group for the X dimension and one for Y dimension.

3. Embedded ABNF

Many IETF protocols define allowable values for their text strings in ABNF [RFC5234] [RFC7405]. It is often desirable to define a text string type in CDDL by employing existing ABNF embedded into the CDDL specification. Without specific ABNF support in CDDL, that ABNF would usually need to be translated into a regular expression (if that is even possible).

ABNF is added to CDDL in the same way that regular expressions were added: by defining a ".abnf" control operator. The target is usually "text" or some restriction on it, the controller is the text of an ABNF specification.

There are several small issues, with solutions given here:

* ABNF can be used to define byte sequences as well as UTF-8 text strings interpreted as Unicode scalar sequences. This means this specification defines two control operators: ".abnfb" for ABNF denoting byte sequences and ".abnf" for denoting sequences of Unicode scalar values (codepoint) represented as UTF-8 text strings. Both control operators can be applied to targets of either string type; the ABNF is applied to sequence of bytes in the string interpreting that as a sequence of bytes (".abnfb") or as a sequence of code points represented as an UTF-8 text string (".abnf"). The controller string MUST be a text string.

* ABNF defines a list of rules, not a single expression (called "elements" in [RFC5234]). This is resolved by requiring the controller string to be one valid "element", followed by zero or more valid "rule" separated from the element by a newline; so the controller string can be built by preceding a piece of valid ABNF by an "element" that selects from that ABNF and a newline.

* For the same reason, ABNF requires newlines; specifying newlines in CDDL text strings is tedious (and leads to essentially unreadable ABNF). The workaround employs the ".cat" operator introduced in Section 2.1 and the syntax for text in byte strings. As is customary for ABNF, the syntax of ABNF itself (NOT the syntax expressed in ABNF!) is relaxed to allow a single linefeed as a newline:

```
CRLF = %x0A / %x0D.0A
```

* One set of rules provided in an ABNF specification is often used in multiple positions, in particular staples such as DIGIT and ALPHA. (Note that all rules referenced need to be defined in each ABNF operator controller string -- there is no implicit import of [RFC5234] Core ABNF or other rules.) The composition this calls for can be provided by the ".cat" operator.

These points are combined into an example in Figure 3, which uses ABNF from [RFC3339] to specify one of the CBOR tags defined in [RFC8943].

```
; for draft-ietf-cbor-date-tag
Tag1004 = #6.1004(text .abnf full-date)
; for RFC 7049
Tag0 = #6.0(text .abnf date-time)

full-date = "full-date" .cat rfc3339
date-time = "date-time" .cat rfc3339

; Note the trick of idiomatically starting with a newline, separating
;   off the element in the .cat from the rule-list
rfc3339 = '
   date-fullyear   = 4DIGIT
   date-month      = 2DIGIT  ; 01-12
   date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on
                             ; month/year
   time-hour       = 2DIGIT  ; 00-23
   time-minute     = 2DIGIT  ; 00-59
   time-second     = 2DIGIT  ; 00-58, 00-59, 00-60 based on leap sec
                             ; rules
   time-secfrac    = "." 1*DIGIT
   time-numoffset  = ("+" / "-") time-hour ":" time-minute
   time-offset     = "Z" / time-numoffset

   partial-time    = time-hour ":" time-minute ":" time-second
                     [time-secfrac]
   full-date       = date-fullyear "-" date-month "-" date-mday
   full-time       = partial-time time-offset

   date-time       = full-date "T" full-time
' .cat rfc5234-core

rfc5234-core = '
         DIGIT          =  %x30-39 ; 0-9
; abbreviated here
'
```

Figure 3: Example: employing RFC 3339 ABNF for defining CBOR Tags

4. Features

Traditionally, the kind of validation enabled by languages such as CDDL provided a Boolean result: valid, or invalid.

In rapidly evolving environments, this is too simplistic. The data models described by a CDDL specification may continually be enhanced by additional features, and it would be useful even for a specification that does not yet describe a specific future feature to identify the extension point the feature can use, accepting such extensions while marking them as such.

The ".feature" control annotates the target as making use of the feature named by the controller. The latter will usually be a string. A tool that validates an instance against that specification may mark the instance as using a feature that is annotated by the specification.

More specifically, the tool's diagnostic output might contain the controller (right hand side) as a feature name, and the target (left hand side) as a feature detail. However, in some cases, the target has too much detail, and the specification might want to hint the tool that more limited detail is appropriate. In this case, the controller should be an array, with the first element being the feature name (that would otherwise be the entire controller), and the second element being the detail (usually another string).

```
foo = {
  kind: bar / baz .feature (["foo-extensions", "bazify"])
}
bar = ...
baz = ... ; complex stuff that doesn't all need to be in the detail
```

Figure 4 shows what could be the definition of a person, with potential extensions beyond "name" and "organization" being marked "further-person-extension". Extensions that are known at the time this definition is written can be collected into "$$person-extensions". However, future extensions would be deemed invalid unless the wildcard at the end of the map is added. These extensions could then be specifically examined by a user or a tool that makes use of the validation result; the label (map key) actually used makes a fine feature detail for the tool's diagnostic output.

Leaving out the entire extension point would mean that instances that make use of an extension would be marked as whole-sale invalid, making the entire validation approach much less useful. Leaving the extension point in, but not marking its use as special, would render mistakes such as using the label "organisation" instead of "organization" invisible.

```
person = {
  ? name: text
  ? organization: text
  $$person-extensions
  * (text .feature "further-person-extension") => any
}

$$person-extensions //= (? bloodgroup: text)
```

Figure 4: Map extensibility with .feature

Figure 5 shows another example where ".feature" provides for type extensibility.

```
allowed-types = number / text / bool / null
              / [* number] / [* text] / [* bool]
              / (any .feature "allowed-type-extension")
```

Figure 5: Type extensibility with .feature

A CDDL tool may simply report the set of features being used; the control then only provides information to the process requesting the validation. One could also imagine a tool that takes arguments allowing the tool to accept certain features and reject others (enable/disable). The latter approach could for instance be used for a JSON/CBOR switch:

```
SenML-Record = {
; ...
  ? v => number
; ...
}
v = JC<"v", 2>
JC<J,C> = J .feature "json" / C .feature "cbor"
```

It remains to be seen if the enable/disable approach can lead to new idioms of using CDDL. The language currently has no way to enforce mutually exclusive use of features, as would be needed in this example.

5. IANA Considerations

This document requests IANA to register the contents of Table 2 into the CDDL Control Operators registry [IANA.cddl]:

Name	Reference
.cat	[RFCthis]
.plus	[RFCthis]
.abnf	[RFCthis]
.abnfb	[RFCthis]
.feature	[RFCthis]

Table 2

6. Implementation Status

An early implementation of the control operator ".feature" has been available in the CDDL tool described in Appendix F of [RFC8610] since version 0.8.11. The validator warns about each feature being used and provides the set of target values used with the feature. ".cat" and ".plus" are also implemented.

Andrew Weiss' [CDDL-RS] has an ongoing implementation of this draft which is feature-complete except for the ABNF support (https://github.com/anweiss/cddl/pull/79 (https://github.com/anweiss/cddl/pull/79)).

7. Security considerations

The security considerations of [RFC8610] apply.

8. References

8.1. Normative References

[IANA.cddl] IANA, "Concise Data Definition Language (CDDL)", <http://www.iana.org/assignments/cddl>.

[RFC2119] 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>.

[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>.

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

[RFC8174] 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>.

[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>.

8.2. Informative References

[CDDL-RS] Weiss, A., "cddl-rs", n.d., <https://github.com/anweiss/cddl>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC8943] Jones, M., Nadalin, A., and J. Richter, "Concise Binary Object Representation (CBOR) Tags for Date", RFC 8943, DOI 10.17487/RFC8943, November 2020, <https://www.rfc-editor.org/info/rfc8943>.

Acknowledgements

Jim Schaad suggested several improvements. The ".feature" feature was developed out of a discussion with Henk Birkholz.

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 5 September 2021

C. Bormann
Universität Bremen TZI

4 March 2021

Additional Control Operators for CDDL
draft-ietf-cbor-cddl-control-03

Abstract

The Concise Data Definition Language (CDDL), standardized in RFC 8610, provides "control operators" as its main language extension point.

The present document defines a number of control operators that did not make it into RFC 8610: ".plus", ".cat" and ".det" for the construction of constants, ".abnf"/".abnfb" for including ABNF (RFC 5234/RFC 7405) in CDDL specifications, and ".feature" for indicating the use of a non-basic feature in an instance.

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 5 September 2021.

Copyright Notice

Table of Contents

1. Introduction

The Concise Data Definition Language (CDDL), standardized in RFC 8610, provides "control operators" as its main language extension point.

The present document defines a number of control operators that did not make it into RFC 8610:

Name	Purpose
.plus	Numeric addition
.cat	String Concatenation
.det	String Concatenation, dedenting rhs
.abnf	ABNF in CDDL (text strings)
.abnfb	ABNF in CDDL (byte strings)
.feature	Detecting feature use in extension points

Table 1: New control operators in this document

1.1. Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses terminology from [RFC8610]. In particular, with respect to control operators, "target" refers to the left hand side operand, and "controller" to the right hand side operand.

2. Computed Literals

CDDL as defined in [RFC8610] does not have any mechanisms to compute literals. As an 80 % solution, this specification adds three control operators: ".plus" for numeric addition, ".cat" for string concatenation, and ".det" for string concatenation with dedenting of the right hand side (controller).

2.1. Numeric Addition

In many cases in a specification, numbers are needed relative to a base number. The ".plus" control identifies a number that is constructed by adding the numeric values of the target and of the controller.

Target and controller MUST be numeric. If the target is a floating point number and the controller an integer number, or vice versa, the sum is converted into the type of the target; converting from a floating point number to an integer selects its floor (the largest integer less than or equal to the floating point number).

```
interval<BASE> = (
  BASE => int              ; lower bound
  (BASE .plus 1) => int    ; upper bound
  ? (BASE .plus 2) => int ; tolerance
)

X = 0
Y = 3
rect = {
  interval<X>
  interval<Y>
}
```

Figure 1: Example: addition to a base value

The example in Figure 1 contains the generic definition of a group "interval" that gives a lower and an upper bound and optionally a tolerance. "rect" combines two of these groups into a map, one group for the X dimension and one for Y dimension.

2.2. String Concatenation

It is often useful to be able to compose string literals out of component literals defined in different places in the specification.

The ".cat" control identifies a string that is built from a concatenation of the target and the controller. As targets and controllers are types, the resulting type is formally the cross-product of the two types, although not all tools may be able to work with non-unique targets or controllers.

Target and controller MUST be strings. The result of the operation has the type of the target. The concatenation is performed on the bytes in both strings. If the target is a text string, the result of that concatenation MUST be valid UTF-8.

```
a = "foo" .cat '
  bar
  baz
'
; on a system where the newline is \n, is the same string as:
b = "foo\n  bar\n  baz\n"
```

Figure 2: Example: concatenation of text and byte string

The example in Figure 2 builds a text string named "a" out of concatenating the target text string ""foo"" and the controller byte string entered in a text form byte string literal. (This particular idiom is useful when the text string contains newlines, which, as shown in the example for "b", may be harder to read when entered in the format that the pure CDDL text string notation inherits from JSON.)

2.3. String Concatenation with Dedenting

Multi-line string literals for various applications, including embedded ABNF (Section 3), need to be set flush left, at least partially. Often, having some indentation in the source code for the literal can promote readability, as in Figure 3.

```
oid = bytes .abnfb ("oid" .det cbor-tags-oid)
roid = bytes .abnfb ("roid" .det cbor-tags-oid)

cbor-tags-oid = '
  oid = 1*arc
  roid = *arc
  arc = [nlsb] %x00-7f
  nlsb = %x81-ff *%x80-ff
'
```

Figure 3: Example: dedenting concatenation

The control operator ".det" works like ".cat", except that the right hand side (controller) is _dedented_ first. For the purposes of this specification, we define dedenting as:

1. determining the smallest amount of left-most white space (number of leading space characters) in all the non-blank lines, and

2. removing exactly that number of leading space characters from each line. For blank (white space only or empty) lines, there may be less (or no) leading space characters than this amount, in which case all leading space is removed.

(The name ".det" is a shortcut for "dedenting cat". The maybe more obvious name ".dedcat" has not been chosen as it is longer and may invoke unpleasant images.)

If left-hand-side (target) dedenting is needed as well, this can be achieved with the slightly longer construct "("" .det lhs) .det rhs".

3. Embedded ABNF

Many IETF protocols define allowable values for their text strings in ABNF [RFC5234] [RFC7405]. It is often desirable to define a text string type in CDDL by employing existing ABNF embedded into the CDDL specification. Without specific ABNF support in CDDL, that ABNF would usually need to be translated into a regular expression (if that is even possible).

ABNF is added to CDDL in the same way that regular expressions were added: by defining a ".abnf" control operator. The target is usually "text" or some restriction on it, the controller is the text of an ABNF specification.

There are several small issues, with solutions given here:

* ABNF can be used to define byte sequences as well as UTF-8 text strings interpreted as Unicode scalar sequences. This means this specification defines two control operators: ".abnfb" for ABNF denoting byte sequences and ".abnf" for denoting sequences of Unicode scalar values (codepoint) represented as UTF-8 text strings. Both control operators can be applied to targets of either string type; the ABNF is applied to sequence of bytes in the string interpreting that as a sequence of bytes (".abnfb") or as a sequence of code points represented as an UTF-8 text string (".abnf"). The controller string MUST be a text string.

* ABNF defines a list of rules, not a single expression (called "elements" in [RFC5234]). This is resolved by requiring the controller string to be one valid "element", followed by zero or more valid "rule" separated from the element by a newline; so the controller string can be built by preceding a piece of valid ABNF by an "element" that selects from that ABNF and a newline.

* For the same reason, ABNF requires newlines; specifying newlines in CDDL text strings is tedious (and leads to essentially unreadable ABNF). The workaround employs the ".cat" operator introduced in Section 2.2 and the syntax for text in byte strings. As is customary for ABNF, the syntax of ABNF itself (NOT the syntax expressed in ABNF!) is relaxed to allow a single linefeed as a newline:

```
CRLF = %x0A / %x0D.0A
```

* One set of rules provided in an ABNF specification is often used in multiple positions, in particular staples such as DIGIT and ALPHA. (Note that all rules referenced need to be defined in each ABNF operator controller string -- there is no implicit import of [RFC5234] Core ABNF or other rules.) The composition this calls for can be provided by the ".cat" operator.

These points are combined into an example in Figure 4, which uses ABNF from [RFC3339] to specify one of the CBOR tags defined in [RFC8943].

```
; for draft-ietf-cbor-date-tag
Tag1004 = #6.1004(text .abnf full-date)
; for RFC 7049
Tag0 = #6.0(text .abnf date-time)

full-date = "full-date" .det rfc3339
date-time = "date-time" .det rfc3339

; Note the trick of idiomatically starting with a newline, separating
;   off the element in the concatenations above from the rule-list
rfc3339 = '
   date-fullyear   = 4DIGIT
   date-month      = 2DIGIT  ; 01-12
   date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on
                             ; month/year
   time-hour       = 2DIGIT  ; 00-23
   time-minute     = 2DIGIT  ; 00-59
   time-second     = 2DIGIT  ; 00-58, 00-59, 00-60 based on leap sec
                             ; rules
   time-secfrac    = "." 1*DIGIT
   time-numoffset  = ("+" / "-") time-hour ":" time-minute
   time-offset     = "Z" / time-numoffset

   partial-time    = time-hour ":" time-minute ":" time-second
                     [time-secfrac]
   full-date       = date-fullyear "-" date-month "-" date-mday
   full-time       = partial-time time-offset

   date-time       = full-date "T" full-time
' .cat rfc5234-core

rfc5234-core = '
   DIGIT          =  %x30-39 ; 0-9
   ; abbreviated here
'
```

Figure 4: Example: employing RFC 3339 ABNF for defining CBOR Tags

4. Features

Traditionally, the kind of validation enabled by languages such as CDDL provided a Boolean result: valid, or invalid.

In rapidly evolving environments, this is too simplistic. The data models described by a CDDL specification may continually be enhanced by additional features, and it would be useful even for a specification that does not yet describe a specific future feature to identify the extension point the feature can use, accepting such extensions while marking them as such.

The ".feature" control annotates the target as making use of the feature named by the controller. The latter will usually be a string. A tool that validates an instance against that specification may mark the instance as using a feature that is annotated by the specification.

More specifically, the tool's diagnostic output might contain the controller (right hand side) as a feature name, and the target (left hand side) as a feature detail. However, in some cases, the target has too much detail, and the specification might want to hint the tool that more limited detail is appropriate. In this case, the controller should be an array, with the first element being the feature name (that would otherwise be the entire controller), and the second element being the detail (usually another string).

```
foo = {
  kind: bar / baz .feature (["foo-extensions", "bazify"])
}
bar = ...
baz = ... ; complex stuff that doesn't all need to be in the detail
```

Figure 5 shows what could be the definition of a person, with potential extensions beyond "name" and "organization" being marked "further-person-extension". Extensions that are known at the time this definition is written can be collected into "$$person-extensions". However, future extensions would be deemed invalid unless the wildcard at the end of the map is added. These extensions could then be specifically examined by a user or a tool that makes use of the validation result; the label (map key) actually used makes a fine feature detail for the tool's diagnostic output.

Leaving out the entire extension point would mean that instances that make use of an extension would be marked as whole-sale invalid, making the entire validation approach much less useful. Leaving the extension point in, but not marking its use as special, would render mistakes such as using the label "organisation" instead of "organization" invisible.

```
person = {
  ? name: text
  ? organization: text
  $$person-extensions
  * (text .feature "further-person-extension") => any
}

$$person-extensions //= (? bloodgroup: text)
```

Figure 5: Map extensibility with .feature

Figure 6 shows another example where ".feature" provides for type extensibility.

```
allowed-types = number / text / bool / null
              / [* number] / [* text] / [* bool]
              / (any .feature "allowed-type-extension")
```

Figure 6: Type extensibility with .feature

A CDDL tool may simply report the set of features being used; the control then only provides information to the process requesting the validation. One could also imagine a tool that takes arguments allowing the tool to accept certain features and reject others (enable/disable). The latter approach could for instance be used for a JSON/CBOR switch:

```
SenML-Record = {
; ...
  ? v => number
; ...
}
v = JC<"v", 2>
JC<J,C> = J .feature "json" / C .feature "cbor"
```

It remains to be seen if the enable/disable approach can lead to new idioms of using CDDL. The language currently has no way to enforce mutually exclusive use of features, as would be needed in this example.

5. IANA Considerations

This document requests IANA to register the contents of Table 2 into the CDDL Control Operators registry [IANA.cddl]:

Name	Reference
.plus	[RFCthis]
.cat	[RFCthis]
.det	[RFCthis]
.abnf	[RFCthis]
.abnfb	[RFCthis]
.feature	[RFCthis]

Table 2

6. Implementation Status

An early implementation of the control operator ".feature" has been available in the CDDL tool described in Appendix F of [RFC8610] since version 0.8.11. The validator warns about each feature being used and provides the set of target values used with the feature. The other control operators defined in this specification are also implemented as of version 0.8.21.

Andrew Weiss' [CDDL-RS] has an ongoing implementation of this draft which is feature-complete except for the ABNF and dedenting support (https://github.com/anweiss/cddl/pull/79 (https://github.com/anweiss/cddl/pull/79)).

7. Security considerations

The security considerations of [RFC8610] apply.

8. References

8.1. Normative References

[IANA.cddl]
IANA, "Concise Data Definition Language (CDDL)", <http://www.iana.org/assignments/cddl>.

[RFC2119] 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>.

[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>.

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

[RFC8174] 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>.

[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>.

8.2. Informative References

[CDDL-RS] Weiss, A., "cddl-rs", n.d., <https://github.com/anweiss/cddl>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC8943] Jones, M., Nadalin, A., and J. Richter, "Concise Binary Object Representation (CBOR) Tags for Date", RFC 8943, DOI 10.17487/RFC8943, November 2020, <https://www.rfc-editor.org/info/rfc8943>.

Acknowledgements

Jim Schaad suggested several improvements. The ".feature" feature was developed out of a discussion with Henk Birkholz. Paul Kyzivat helped isolate the need for ".det".

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

CBOR Working Group
Internet-Draft
Intended status: Best Current Practice
Expires: 23 October 2021

M. Richardson
Sandelman Software Works

21 April 2021

On storing CBOR encoded items on stable storage
draft-ietf-cbor-file-magic-01

Abstract

This document proposes an on-disk format for CBOR objects that is friendly to common on-disk recognition systems like the Unix file(1) command.

This document is being discussed at: https://github.com/cbor-wg/cbor-magic-number

Table of Contents

1. Introduction

Since very early in computing, operating systems have sought ways to mark which files could be processed by which programs.

For instance, the Unix file(1) command, which has existed since 1973 [file], has been able to identify many file formats for decades. Many systems (Linux, MacOS, Windows) will select the correct application based upon the file contents, if the system can not determine it by other means: for instance, the classic MacOS maintained a resource fork that includes media type ("MIME type") information and therefore ideally never needs to know what anything about the file. Other systems do this by file extensions.

While having a media type associated with the file is a better solution in general, when files become disconnected from their type information, such as when attempting to do forensics on a damaged system, then being able to identify a file type can become very important.

It is noted that in the media type registration, that a magic number is asked for, if available, as is a file extension.

A challenge for the file(1) program is often that it can be confused by the encoding vs the content. For instance, an Android "apk" used to transfer and store an application may be identified as a ZIP file. Both OpenOffice or MSOffice files are ZIP files of XML files. (Unless OpenOffice files are flat (fodp) files, in which case they may appear to be generic XML files.)

As CBOR becomes a more and more common encoding for a wide variety of artifacts, identifying them as just "CBOR" is probably not sufficient. This document provides a way to encode a magic number into the beginning of a CBOR format file. Two options are presented: typically a CBOR Protocol author will specify one.

A CBOR Protocol is a specification which uses CBOR as its encoding. Examples of CBOR Protocols currently under development include CoSWID [I-D.ietf-sacm-coswid], and EAT [I-D.ietf-rats-eat]. COSE itself [RFC8152] is considered infrastructure, however the encoding of public keys in CBOR as described in [I-D.mattsson-cose-cbor-cert-compress] would be an identified CBOR Protocol.

A major inspiration for this document is observing the mess in ASN.1 based systems where most files are PEM encoded, identified by the extension "pem", confusing public keys, private keys, certificate requests and SIME content.

These proposals are invasive to how CBOR protocols are written to disk, but in both cases, the proposed envelope does not require that the tag be transfered on the wire.

In addition to the on-disk identification aspects, there are some protocols which may benefit from having such a magic number on the wire if they presently using a different (legacy) encoding scheme. The presence of the identifiable magic sequence signals that CBOR is being used or a legacy scheme.

2. Requirements for a Magic Number

A magic number is ideally a unique fingerprint, present in the first 4 or 8 bytes of the file, which does not change when the contents change, and does not depend upon the length of the file.

Less ideal solutions have a pattern that needs to be matched, but in which some bytes need to be ignored. While the Unix file(1) command can be told to ignore bytes, this can lead to ambiguities.

3. Protocol

There are two variations of this practice. Both use CBOR Tags in a way that results in a deterministic first 8 to 12 bytes.

3.1. The CBOR Protocol Specific Tag

CBOR Protocol designers should obtain a tag for each major type of object that they might store on disk. As there are more than 4 million available 4-byte tags, there should be little issue in allocating a few to each available CBOR Protocol.

The policy is First Come First Served, so all that is required is an email to IANA, having filled in the small template provided in section 9.2 of [RFC8949].

This tag should be allocated by the author of the CBOR Protocol, and to be in the four-byte range, it should be at least 0x01000000 (decimal 16777216) in value.

The use of a sequence of four US-ASCII codes which are mnemonic to the protocol is encouraged, but not required.

3.2. CBOR Tag Wrapped

This proposal starts with the Self-described CBOR tag, 55799, as described in [RFC8949] section 3.4.6.

A second CBOR Tag is then allocated to describe the specific Protocol involved, as described above.

This proposal wraps the CBOR value as tags usually do. Applications that need to send the CBOR value across a constrained link may wish to remove the two tags if the use is implicitly understood. This is a decision of the CBOR Protocol specification.

3.3. CBOR Tag Sequence

This proposal makes use of CBOR Sequences as described in [RFC8742].

This proposal consists of two tags and a constant string for a total of 12 bytes.

1. The file shall start with the Self-described CBOR Sequence tag, 55800.

2. The file shall continue with a CBOR tag, from the First Come First Served space, which uniquely identifies the CBOR Protocol. The use of a four-byte tag is encouraged.

3. The three byte CBOR byte string containing 0x42_4F_52. When encoded it shows up as "CBOR"

The first part identifies the file as being CBOR, and does so with all the desirable properties explained in [RFC8949] section 3.4.6. Specifically, it does not seem to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The second part identifies which CBOR Protocol is used, as described above.

The third part is a constant value 0x43_42_4f_52, "CBOR". That is, it the three byte sequence 0x42_4f_52 ("BOR"). This is the data item that is tagged.

The actual CBOR Protocol value then follows as the next data item(s) in the CBOR sequence, without a need for any further specific tag. The use of a CBOR Sequence allows the application to trivially remove the first item with the two tags.

This means that should a file be reviewed by a human (directly in an editor, or in a hexdump display), it will include the string "CBOR" prominently. This value is also included simply because the two tags need to tag something.

4. Security Considerations

This document provides a way to identify CBOR Protocol objects. Clearly identifying CBOR contents on disk may have a variety of impacts.

The most obvious is that it may allow malware to identify interesting objects on disk, and then corrupt them.

5. IANA Considerations

There are no IANA actions. This section documents the allocation that was done.

5.1. CBOR Sequence Tag

IANA has allocated tag 55800 as the CBOR Sequence tag. This tag is from the First Come/First Served area.

The value has been picked to have properties similiar to the 55799 tag.

The hexadecimal representation is: 0xd9_\d9_f8.

This is not valid UTF-8: the first 0xd9 puts the value into the three-byte value of UTF-8, but the 0xd9 as the second value is not a valid second byte for UTF-8.

This is not valid UTF-16: the byte sequence 0xd9d9 (in either endian order), puts this value into the UTF-16 high-half zone, which would signal that this a 32-bit Unicode value. However, the following 16-bit big-endian value 0xf8.. is not a valid second sequence according to [RFC2781]. On a little-endian system, it would be necessary to examine the fourth byte to determine if it is valid. That next byte is determined by the subsequent encoding, and [RFC8949] section 3.4.6 has already determined that no valid CBOR encodings result in a valid UTF-16.

Data Item: byte string
Semantics: indicates that the file contains CBOR Sequences

6. Acknowledgements

The CBOR WG brainstormed this protocol on January 20, 2021.

7. Changelog

8. References

8.1. Normative References

[BCP14] 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>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC8949] 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>.

8.2. Informative References

[file] Wikipedia, "file (command)", 20 January 2021, <https://en.wikipedia.org/wiki/File_%28command%29>.

[I-D.ietf-rats-eat] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-09, 7 March 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-09.txt>.

[I-D.ietf-sacm-coswid] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-17, 22 February 2021, <https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-17.txt>.

[I-D.mattsson-cose-cbor-cert-compress] Raza, S., Höglund, J., Selander, G., Mattsson, J. P., and M. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-mattsson-cose-cbor-cert-compress-08, 22 February 2021, <https://www.ietf.org/archive/id/draft-mattsson-cose-cbor-cert-compress-08.txt>.

[ilbm] Wikipedia, "Interleaved BitMap", 20 January 2021, <https://en.wikipedia.org/wiki/ILBM>.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, DOI 10.17487/RFC2781, February 2000, <https://www.rfc-editor.org/info/rfc2781>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

Appendix A. Example from Openswan

The Openswan IPsec project has a daemon ("pluto"), and two control programs ("addconn", and "whack"). They communicate via a Unix-domain socket, over which a C-structure containing pointers to strings is serialized using a bespoke mechanism. This is normally not a problem as the structure is compiled by the same compiler; but when there are upgrades it is possible for the daemon and the control programs to get out of sync by the bespoke serialization. As a result, there are extra compensations to deal with shutting the daemon down. During testing it is sometimes the case that upgrades are backed out.

In addition, when doing unit testing, the easiest way to load policy is to use the normal policy reading process, but that is not normally loaded in the daemon. Instead the IPC that is normally sent across the wire is compiled/serialized and placed in a file. The above magic number is included in the file, and also on the IPC in order to distinguish the "shutdown" command CBOR operation.

In order to reduce the problems due to serialization, the serialization is being changed to CBOR. Additionally, this change allows the IPC to be described by CDDL, and for any language that encode to CBOR can be used.

IANA has allocated the tag 1330664270, or 0x4f_50_ 53_ 4e for this purpose. As a result, each file and each IPC is prefixed with:

In diagnostic notation: ~~~~ 55800(1330664270(h'424F52')) ~~~~

Or in hex: ~~~~ 00000000 d9 d9 f9 da 4f 50 53 4e 43 42 4f 52 |....OPSNCBOR| ~~~~

Contributors

Carsten Bormann

Email: cabo@tzi.org

Josef 'Jeff' Sipek

Email: jeffpc@josefsipek.net

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

CBOR Working Group
Internet-Draft
Intended status: Standards Track
Expires: 23 October 2021

M. Richardson
Sandelman Software Works
21 April 2021

CBOR tags for IPv4 and IPv6 addresses and prefixes
draft-ietf-cbor-network-addresses-04

Abstract

This document describes two CBOR Tags to be used with IPv4 and IPv6 addresses and prefixes.

RFC-EDITOR-please remove: This work is tracked at https://github.com/cbor-wg/cbor-network-address

Table of Contents

1. Introduction

[RFC8949] defines a number of CBOR Tags for common items.

Not included are ones to indicate if the item is an IPv4 or IPv6 address, or if it is an address plus prefix length. This document defines them.

2. Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Protocol

These tags can applied to byte strings to represent a single address.

When applied to an array, the represent a CIDR-style prefix. When a byte string (without prefix) appears in a context where a prefix is expected, then it is to be assumed that all bits are relevant. That is, for IPv4, a /32 is implied, and for IPv6, a /128 is implied.

3.1. IPv6

IANA has allocated tag 54 for IPv6 uses. (Note that this is the ASCII code for '6'.)

An IPv6 address is to be encoded as a sixteen-byte byte string ([RFC8949] section, 3.1, major type 2), prefixed with Tag(54).

An IPv6 prefix, such as 2001:db8:1234::/48 is to be encoded as a two element array, with the length of the prefix first. Trailing zero bytes MUST be omitted.

For example:

54([48, h'20010db81234'])

3.2. IPv4

IANA has allocated tag 54 for IPv4 uses. (Note that this is the ASCII code for '4'.)

An IPv4 address is to be encoded as a four-byte byte string ([RFC8949] section, 3.1, major type 2), prefixed with Tag(52).

An IPv4 prefix, such as 192.0.2.1/24 is to be encoded as a two element array, with the length of the prefix first. Trailing zero bytes MUST be omitted.

For example:

52([24, h'C00002'])

4. Encoder Consideration for prefixes

An encoder may omit as many right-hand (trailing) bytes which are all zero as it wishes.

There is no relationship between the number of bytes omitted and the prefix length. For instance, the prefix 2001:db8::/64 is optimally encoded as:

54([64, h'20010db8'])

An encoder MUST take care to set all trailing bits to zero. While decoders are expected to ignore them, such garbage entities could be used as a covert channel, or may reveal the state of what would otherwise be private memory contents. So for example, 2001:db8:1230::/44 MUST be encoded as:

52([44, h'20010db81230'])

even though variations like:

```
54([44, h'20010db81233'])  WRONG
54([45, h'20010db8123f'])  WRONG
```

would be parsed in the exact same way.

The same considerations apply to IPv4 prefixes.

5. Decoder Considerations for prefixes

A decoder MUST consider all bits to the right of the prefix length to be zero.

A decoder MUST handle the case where a prefix length specifies that more bits are relevant than are actually present in the byte-string. As a pathological case, ::/128 can be encoded as

```
54([128, h''])
```

A recommendation for implementation is to first create an array of 16 (or 4) bytes in size, set it all to zero.

Then looking at the length of the included byte-string, and of the prefix-length, rounded up to the next multiple of 8, and taking whichever is smaller, copy that many bytes from the byte-string into the array.

Finally, looking at the last three bits of the prefix-length (that is, the prefix-length modulo 8), use a static array of 8 values to force the lower bits, non-relevant bits to zero.

A particularly paranoid decoder could examine the lower non-relevant bits to determine if they are non-zero, and reject the prefix. This would detect non-compliant encoders, or a possible covert channel.

6. Security Considerations

Identifying which byte sequences in a protocol are addresses may allow an attacker or eavesdropper to better understand what parts of a packet to attack.

Reading the relevant RFC may provide more information, so it would seem that any additional security that was provided by not being able to identify what are IP addresses falls into the security by obscurity category.

The right-hand bits of the prefix, after the prefix-length, are ignored by this protocol. A malicious party could use them to transmit covert data in a way that would not affect the primary use

of this encoding. Such abuse would be detected by examination of the raw protocol bytes. Users of this encoding should be aware of this possibility.

7. IANA Considerations

IANA has allocated two tags from the Specification Required area of the Concise Binary Object Representation (CBOR) Tags:

7.1. Tag 54 - IPv6

Data Item: byte string or array
Semantics: IPv6 or [prefixlen,IPv6]

7.2. Tag 52 - IPv4

Data Item: byte string or array
Semantics: IPv4 or [prefixlen,IPv4]

8. Normative References

[RFC2119] 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>.

[RFC8174] 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>.

[RFC8949] 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>.

Appendix A. Changelog

This section is to be removed before publishing as an RFC.

* 03

* 02

* 01 added security considerations about covert channel

Acknowledgements

none yet

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 13 August 2021

C. Bormann
Universität Bremen TZI

9 February 2021

Packed CBOR
draft-ietf-cbor-packed-02

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

CBOR does not provide any forms of data compression. CBOR data items, in particular when generated from legacy data models often allow considerable gains in compactness when applying data compression. While traditional data compression techniques such as DEFLATE (RFC 1951) work well for CBOR, their disadvantage is that the receiver needs to unpack the compressed form to make use of data.

This specification describes Packed CBOR, a simple transformation of a CBOR data item into another CBOR data item that is almost as easy to consume as the original CBOR data item. A separate decompression step is therefore often not required at the receiver.

Note to Readers

This is a working-group draft of the CBOR working group of the IETF, https://datatracker.ietf.org/wg/cbor/about/ (https://datatracker.ietf.org/wg/cbor/about/). Discussion takes places on the github repository https://github.com/cbor-wg/cbor-packed (https://github.com/cbor-wg/cbor-packed) and on the CBOR WG mailing list, https://www.ietf.org/mailman/listinfo/cbor (https://www.ietf.org/mailman/listinfo/cbor).

Status of This Memo

Table of Contents

1. Introduction

(TO DO, expand on text from abstract here; move references here and neuter them in the abstract as per Section 4.3 of [RFC7322].)

The specification defines a transformation from a Packed CBOR data item to the original CBOR data item; it does not define an algorithm for an actual packer. Different packers can differ in the amount of effort they invest in arriving at a minimal packed form.

Packed CBOR can employ two kinds of optimization:

* item sharing: substructures (data items) that occur repeatedly in the original CBOR data item can be collapsed to a simple reference to a common representation of that data item. The processing required during consumption is limited to following that reference.

* affix sharing: data items (strings, containers) that share a prefix or suffix (affix) can be replaced by a reference to a common affix plus the rest of the data item. For strings, the processing required during consumption is similar to following the affix reference plus that for an indefinite-length string.

A specific application protocol that employs Packed CBOR might allow both kinds of optimization or limit the representation to item sharing only.

1.1. Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Packed reference: A shared item reference or an affix reference

Shared item reference: A reference to a shared item as defined in Section 2.2

Affix reference: A reference that combines an affix item as defined in Section 2.3.

Affix: Prefix or suffix.

Packing tables: The triple of a shared item table, a prefix table, and a suffix table.

Expansion: The result of applying a packed reference in the context of given Packing tables.

The definitions of [RFC8949] apply. The term "byte" is used in its now customary sense as a synonym for "octet". Where bit arithmetic is explained, this document uses the notation familiar from the programming language C (including C++14's 0bnnn binary literals), except that, in the plain text form of this document, the operator "^" stands for exponentiation.

2. Packed CBOR

Packed CBOR is defined in two parts: Referencing packing tables (this section) and setting up packing tables (Section 3).

2.1. Packing Tables

At any point within a data item making use of Packed CBOR, there is a Current Set of packing tables that applies.

There are three packing tables in a Current Set:

* Shared item table
* Prefix table
* Suffix table

Without any table setup, all these tables are empty arrays. Table setup can cause these arrays to be non-empty, where the elements are (potentially themselves packed) data items. In the abstract, each of the tables is indexed by an unsigned integer (starting from 0).

2.2. Referencing Shared Items

Shared items are stored in the shared item table of the Current Set.

The shared data items are referenced by using the reference data items in Table 1. When reconstructing the original data item, such a reference is replaced by the referenced data item, which is then recursively unpacked.

reference	table index
Simple value 0-15	0-15
Tag 6(unsigned integer N)	16 + 2*N
Tag 6(negative integer N)	16 - 2*N - 1

Table 1: Referencing Shared Values

As examples in CBOR diagnostic notation (Section 8 of [RFC8949]), the first 22 elements of the shared item table are referenced by "simple(0)", "simple(1)", ... "simple(15)", "6(0)", "6(-1)", "6(1)", "6(-2)", "6(2)", "6(-3)". (The alternation between unsigned and negative integers for even/odd table index values makes systematic use of shorter integer encodings first.)

Taking into account the encoding of these referring data items, there are 16 one-byte references, 48 two-byte references, 512 three-byte references, 131072 four-byte references, etc. As integers can grow to very large (or negative) values, there is no practical limit to how many shared items might be used in a Packed CBOR item.

Note that the semantics of Tag 6 depend on its content: An integer turns the tag into a shared item reference, a string or container (map or array) into a prefix reference (see Table 2).

2.3. Referencing Affix Items

Prefix items are stored in the prefix table of the Current Set; suffix items are stored in the suffix table of the Current Set. We collectively call these items affix items; when referencing, which of the tables is actually used depends on whether a prefix or a suffix reference was used.

prefix reference	table index
Tag 6(suffix)	0
Tag 225-255(suffix)	1-31
Tag 28704-32767(suffix)	32-4095
Tag 1879052288-2147483647(suffix)	4096-268435455

Table 2: Referencing Prefix Values

suffix reference	table index
Tag 216-223(prefix)	0-7
Tag 27647-28671(prefix)	8-1023
Tag 1811940352-1879048191(prefix)	1024-67108863

Table 3: Referencing Suffix Values

Affix data items are referenced by using the data items in Table 2 and Table 3. Each of these implies the table used (prefix or suffix), a table index (an unsigned integer) and contains a "rump item". When reconstructing the original data item, such a reference is replaced by a data item constructed from the referenced affix data item (affix, which might need to be recursively unpacked first) "concatenated" with the tag content (rump, again possibly recursively unpacked).

* For a rump of type array and map, the affix also needs to be an array or a map. For an array, the elements from the prefix are prepended, and the elements from a suffix are appended to the rump array. For a map, the entries in the affix are added to those of the rump; prefix and suffix references differ in how entries with identical keys are combined: for prefix references, an entry in the rump with the same key as an entry in the affix overrides the one in the affix, while for suffix references, an entry in the affix overrides an entry in the rump that has the same key.

> ISSUE: Not sure that we want to use the efficiencies of overriding, but having default values supplied out of a dictionary to be overridden by a rump sounds rather handy. Note that there is no way to remove a map entry from the table.

* For a rump of one of the string types, the affix also needs to be one of the string types; the bytes of the strings are concatenated as specified (prefix + rump, rump + suffix). The result of the concatenation gets the type of the rump; this way a single affix can be used to build both byte and text strings, depending on what type of rump is being used.

As a contrived (but short) example, if the prefix table is "["foobar", "foob", "fo"]", the following prefix references will all unpack to ""foobart"": "6("t")", "224("art")", "225("obart")" (the last example isn't really an optimization).

Taking into account the encoding, there is one single-byte prefix reference, 31 (2^(5)-2^(0)) two-byte references, 4064 (2^(12)-2^(5)) three-byte references, and 26843160 (2^(28)-2^(12)) five-byte references for prefixes. 268435455 (2^(28)) is an artificial limit, but should be high enough that there, again, is no practical limit to how many prefix items might be used in a Packed CBOR item. The numbers for suffix references are one quarter of those, except that there is no single-byte reference and 8 two-byte references.

> Rationale: Experience suggests that prefix packing might be more likely than suffix packing. Also for this reason, there is no intent to spend a 1+0 tag value for suffix matching.

2.4. Discussion

This specification uses up a large number of Simple Values and Tags, in particular one of the rare one-byte tags and half of the one-byte simple values. Since the objective is compression, this is warranted if and only if there is consensus that this specific format could be useful for a wide area of applications, while maintaining reasonable simplicity in particular at the side of the consumer.

A maliciously crafted Packed CBOR data item might contain a reference loop. A consumer/decompressor MUST protect against that.

The current definition does nothing to help with packing CBOR sequences [RFC8742]; maybe it should.

3. Table Setup

The packing references described in Section 2 assume that packing tables have been set up.

By default, all three tables are empty (zero-length arrays).

Table setup can happen in one of two ways:

* By the application environment, e.g., a media type. These can define tables that amount to a static dictionary that can be used in a CBOR data item for this application environment.

* By one or more tags enclosing the packed content. These can be defined to add to the packing tables that already apply to the tag. Usually, the semantics of the tag will be to prepend items to one of the tables. Note that it may be useful to leave a particular efficiency tier alone and only prepend to a higher tier; e.g., a tag could insert shared items at table index 16 and shift anything that was already there further down in the array while leaving index 0 to 15 alone. Explicit additions by tag can combine with application-environment supplied tables that apply to the entire CBOR data item.

The present specification only defines a single tag for prepending to the (by default empty) tables.

> We could also define a tag for dictionary referencing (or include that in the basic packed CBOR), but the details are likely to vary considerably between applications. A URI-based reference would be easy to add, but might be too inefficient when used in the likely combination with an "ni:" URI [RFC6920].

3.1. Basic Packed CBOR

A predefined tag for packing table setup is defined in CDDL [RFC8610] as in Figure 1:

```
Basic-Packed-CBOR = #6.51([[*shared], [*prefix], [*suffix], rump])
rump = any
prefix = any
suffix = any
shared = any
```

Figure 1: Packed CBOR in CDDL

(This assumes the allocation of tag number 51.)

The arrays given as the first, second, and third element of the content of the tag 51 are prepended to the tables for shared items, prefixes, and suffixes that apply to the entire tag (by default empty tables).

The original CBOR data item can be reconstructed by recursively replacing shared, prefix, and suffix references encountered in the rump by their expansions.

4. IANA Considerations

In the registry [IANA.cbor-tags], IANA is requested to allocate the tags defined in Table 4.

Tag	Data Item	Semantics	Reference
6	integer, text string, byte string, array, map	Packed CBOR: shared/ prefix	draft-ietf-cbor-packed
225-255	text string, byte string, array, map	Packed CBOR: prefix	draft-ietf-cbor-packed
28704-32767	text string, byte string, array, map	Packed CBOR: prefix	draft-ietf-cbor-packed
1879052288-2147483647	text string, byte string, array, map	Packed CBOR: prefix	draft-ietf-cbor-packed
216-223	text	Packed	draft-ietf-cbor-packed

	string, byte string, array, map	CBOR: suffix	
27647-28671	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed
1811940352-1879048191	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed

Table 4: Values for Tag Numbers

In the registry [IANA.cbor-simple-values], IANA is requested to allocate the simple values defined in Table 5.

Value	Semantics	Reference
0-15	Packed CBOR: shared	draft-ietf-cbor-packed

Table 5: Simple Values

5. Security Considerations

The security considerations of [RFC8949] apply.

Loops in the Packed CBOR can be used as a denial of service attack, see Section 2.4.

As the unpacking is deterministic, packed forms can be used as signing inputs. (Note that if external dictionaries are added to cbor-packed, this requires additional consideration.)

6. References

6.1. Normative References

[IANA.cbor-simple-values]
IANA, "Concise Binary Object Representation (CBOR) Simple Values", <http://www.iana.org/assignments/cbor-simple-values>.

[IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags>.

[RFC2119] 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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8174] 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>.

[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>.

[RFC8949] 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

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <https://www.rfc-editor.org/info/rfc7322>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

Appendix A. Examples

The (JSON-compatible) CBOR data structure depicted in Figure 2, 400 bytes of binary CBOR, could lead to a packed CBOR data item depicted in Figure 3, ~309 bytes. Note that this particular example does not lend itself to prefix compression.

```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

Figure 2: Example original CBOR data item

```
   51(["price", "category", "author", "title", "fiction", 8.95, "isbn"],
      /  0          1         2         3         4       5      6   /
      [], [],
      [{"store": {
         "book": [
           {simple(1): "reference", simple(2): "Nigel Rees",
            simple(3): "Sayings of the Century", simple(0): simple(5)},
           {simple(1): simple(4), simple(2): "Evelyn Waugh",
            simple(3): "Sword of Honour", simple(0): 12.99},
           {simple(1): simple(4), simple(2): "Herman Melville",
            simple(3): "Moby Dick", simple(6): "0-553-21311-3",
            simple(0): simple(5)},
           {simple(1): simple(4), simple(2): "J. R. R. Tolkien",
            simple(3): "The Lord of the Rings",
            simple(6): "0-395-19395-8", simple(0): 22.99}],
         "bicycle": {"color": "red", simple(0): 19.95}}}])
```

Figure 3: Example packed CBOR data item

The (JSON-compatible) CBOR data structure below has been packed with shared item and (partial) prefix compression only.

```
{
  "name": "MyLED",
  "interactions": [
    {
      "links": [
        {
          "href": "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueRed",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueRed",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "href": "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueGreen",
          "mediaType": "application/json"
```

```
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueGreen",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "href": "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueBlue",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueBlue",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "href": "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueWhite",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueWhite",
      "writable": true,
      "@type": [
        "Property"
      ]
```

```
    },
    {
      "links": [
        {
          "href": "http://192.168.1.103:8445/wot/thing/MyLED/ledOnOff",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "boolean"
        }
      },
      "name": "ledOnOff",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "href": "http://192.168.1.103:8445/wot/thing/MyLED/colorTemperatureChan
ged",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "colorTemperatureChanged",
      "@type": [
        "Event"
      ]
    }
  ],
  "@type": "Lamp",
  "id": "0",
  "base": "http://192.168.1.103:8445/wot/thing",
  "@context": "http://192.168.1.102:8444/wot/w3c-wot-td-context.jsonld"
}
```

Figure 4: Example original CBOR data item

```
 51([/shared/["name", "@type", "links", "href", "mediaType",
             /  0       1       2        3         4 /
     "application/json", "outputData", {"valueType": {"type":
          /  5              6               7 /
     "number"}}, ["Property"], "writable", "valueType", "type"],
                /   8            9           10           11 /
    /prefix/ ["http://192.168.1.10", 6("3:8445/wot/thing"),
               / 6                        225 /
    225("/MyLED/"), 226("rgbValue"), "rgbValue",
      / 226             227           228     /
    {simple(6): simple(7), simple(9): true, simple(1): simple(8)}],
      / 229 /
    /suffix/ [],
    /rump/ {simple(0): "MyLED",
            "interactions": [
      229({simple(2): [{simple(3): 227("Red"), simple(4): simple(5)}],
       simple(0): 228("Red")}),
      229({simple(2): [{simple(3): 227("Green"), simple(4): simple(5)}],
       simple(0): 228("Green")}),
      229({simple(2): [{simple(3): 227("Blue"), simple(4): simple(5)}],
       simple(0): 228("Blue")}),
      229({simple(2): [{simple(3): 227("White"), simple(4): simple(5)}],
       simple(0): "rgbValueWhite"}),
      {simple(2): [{simple(3): 226("ledOnOff"), simple(4): simple(5)}],
       simple(6): {simple(10): {simple(11): "boolean"}}, simple(0):
       "ledOnOff", simple(9): true, simple(1): simple(8)},
      {simple(2): [{simple(3): 226("colorTemperatureChanged"),
       simple(4): simple(5)}], simple(6): simple(7), simple(0):
       "colorTemperatureChanged", simple(1): ["Event"]}],
      simple(1): "Lamp", "id": "0", "base": 225(""),
      "@context": 6("2:8444/wot/w3c-wot-td-context.jsonld")}])
```

Figure 5: Example packed CBOR data item

Acknowledgements

CBOR packing was originally invented with the rest of CBOR, but did not make it into [RFC7049], the predecessor of [RFC8949]. Various attempts to come up with a specification over the years didn't proceed. In 2017, Sebastian Käbisch proposed investigating compact representations of W3C Thing Descriptions, which prompted the author to come up with essentially the present design.

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440

D-28359 Bremen
Germany

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 23 August 2021

C. Bormann
Universität Bremen TZI

19 February 2021

Concise Binary Object Representation (CBOR) Tags for Object Identifiers
draft-ietf-cbor-tags-oid-05

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

The present document defines CBOR tags for object identifiers (OIDs). It is intended as the reference document for the IANA registration of the CBOR tags so defined.

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 23 August 2021.

Copyright Notice

Table of Contents

1. Introduction

The Concise Binary Object Representation (CBOR, [RFC8949]) provides for the interchange of structured data without a requirement for a pre-agreed schema. [RFC8949] defines a basic set of data types, as well as a tagging mechanism that enables extending the set of data types supported via an IANA registry.

The present document defines CBOR tags for object identifiers (OIDs, [X.660]), which many IETF protocols carry. The ASN.1 Basic Encoding Rules (BER, [X.690]) specify binary encodings of both (absolute) object identifiers and relative object identifiers. The contents of these encodings (the "value" part of BER's type-length-value structure) can be carried in a CBOR byte string. This document defines two CBOR tags that cover the two kinds of ASN.1 object identifiers encoded in this way. The tags can also be applied to arrays and maps to efficiently tag all elements of an array or all keys of a map. It is intended as the reference document for the IANA registration of the tags so defined.

1.1. Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The terminology of [RFC8949] applies; in particular the term "byte" is used in its now customary sense as a synonym for "octet". The term "SDNV" is used as defined in [RFC6256].

2. Object Identifiers

The International Object Identifier tree [X.660] is a hierarchically managed space of identifiers, each of which is uniquely represented as a sequence of unsigned integer values [X.680]. (These integer values are called "primary integer values" in X.660 because they can be accompanied by (not necessarily unambiguous) secondary identifiers. We ignore the latter and simply use the term "integer values" here, occasionally calling out their unsignedness. We also use the term "arc" when the focus is on the edge of the tree labeled by such an integer value, as well as in the sense of a "long arc", i.e. a (sub)sequence of such integer values.)

While these sequences can easily be represented in CBOR arrays of unsigned integers, a more compact representation can often be achieved by adopting the widely used representation of object identifiers defined in BER; this representation may also be more amenable to processing by other software that makes use of object identifiers.

BER represents the sequence of unsigned integers by concatenating self-delimiting [RFC6256] representations of each of the integer values in sequence.

ASN.1 distinguishes absolute object identifiers (ASN.1 Type "OBJECT IDENTIFIER"), which begin at a root arc ([X.660] Clause 3.5.21), from relative object identifiers (ASN.1 Type "RELATIVE-OID"), which begin relative to some object identifier known from context ([X.680] Clause 3.8.63). As a special optimization, BER combines the first two integers in an absolute object identifier into one numeric identifier by making use of the property of the hierarchy that the first arc has only three integer values (0, 1, and 2), and the second arcs under 0 and 1 are limited to the integer values between 0 and 39. (The root arc "joint-iso-itu-t(2)" has no such limitations on its second arc.) If X and Y are the first two integer values, the single integer value actually encoded is computed as:

X * 40 + Y

The inverse transformation (again making use of the known ranges of X and Y) is applied when decoding the object identifier.

Since the semantics of absolute and relative object identifiers differ, this specification defines two tags, collectively called the "OID tags" here:

Tag TBD111: tags a byte string as the [X.690] encoding of an absolute object identifier (simply "object identifier" or "OID").

Tag TBD110: tags a byte string as the [X.690] encoding of a relative object identifier (also "relative OID"). Since the encoding of each number is the same as for [RFC6256] Self-Delimiting Numeric Values (SDNVs), this tag can also be used for tagging a byte string that contains a sequence of zero or more SDNVs.

Tag TBD112: structurally like TBD110, but understood to be relative to "1.3.6.1.4.1" (IANA Private Enterprise Number OID, [IANA.enterprise-numbers]). Hence, the semantics of the result are that of an absolute object identifier.

2.1. Requirements on the byte string being tagged

To form a valid tag, a byte string tagged by TBD111, TBD110, or TBD112 MUST be a syntactically valid BER representation of an object identifier: A concatenation of zero or more SDNV values, where each SDNV value is a sequence of one or more bytes that all have their most significant bit set, except for the last byte, where it is unset. Also, the first byte of each SDNV cannot be a leading zero in SDNV's base-128 arithmetic, so it cannot take the value 0x80 (bullet (c) in Section 8.1.2.4.2 of [X.690]).

In other words:

* the byte string's first byte, and any byte that follows a byte that has the most significant bit unset, MUST NOT be 0x80 (this requirement requires expressing the integer values in their shortest form, with no leading zeroes)

* the byte string's last byte MUST NOT have the most significant bit set (this requirement excludes an incomplete final integer value)

If either of these invalid conditions are encountered, the tag is invalid.

[X.680] restricts RELATIVE-OID values to have at least one arc, i.e., their encoding would have at least one SDNV. This specification permits empty relative object identifiers; they may still be excluded by application semantics.

To facilitate the search for specific object ID values, it is RECOMMENDED that definite length encoding (see Section 3.2.3 of [RFC8949]) is used for the byte strings used as tag content for these tags.

The valid set of byte strings can also be expressed using regular expressions on bytes, using no specific notation but resembling [PCRE]. Unlike typical regular expressions that operate on character sequences, the following regular expressions take bytes as their domain, so they can be applied directly to CBOR byte strings.

For byte strings with tag TBD111:

```
"/^(([\x81-\xFF][\x80-\xFF]*)?[\x00-\x7F])+$/"
```

For byte strings with tag TBD110 or TBD112:

```
"/^(([\x81-\xFF][\x80-\xFF]*)?[\x00-\x7F])*$/"
```

A tag with tagged content that does not conform to the applicable regexp is invalid.

2.2. Discussion

Staying close to the way object identifiers are encoded in ASN.1 BER makes back-and-forth translation easy; otherwise we would choose a more efficient encoding. Object identifiers in IETF protocols are serialized in dotted decimal form or BER form, so there is an advantage in not inventing a third form. Also, expectations of the cost of encoding object identifiers are based on BER; using a different encoding might not be aligned with these expectations. If additional information about an OID is desired, lookup services such as the OID Resolution Service (ORS) [X.672] and the OID Repository [OID-INFO] are available.

3. Basic Examples

This section gives simple examples of an absolute and a relative object identifier, represented via tag number TBD111 and TBD110, respectively.

RFC editor: These and other examples assume the allocation of 111 for TBD111 and 110 for TBD110 and need to be changed if that isn't the actual allocation. Please remove this paragraph.

3.1. Encoding of the SHA-256 OID

ASN.1 Value Notation: { joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) sha256(1) }

Dotted Decimal Notation: 2.16.840.1.101.3.4.2.1

```
06                                  # UNIVERSAL TAG 6
   09                               # 9 bytes, primitive
      60 86 48 01 65 03 04 02 01    # X.690 Clause 8.19
#      |   840  1  |  3  4  2  1     show component encoding
#   2.16         101
```

Figure 1: SHA-256 OID in BER

```
D8 6F                               # tag(111)
   49                               # 0b010_01001: mt 2, 9 bytes
      60 86 48 01 65 03 04 02 01    # X.690 Clause 8.19
```

Figure 2: SHA-256 OID in CBOR

3.2. Encoding of a MIB Relative OID

Given some OID (e.g., "lowpanMib", assumed to be "1.3.6.1.2.1.226" [RFC7388]), to which the following is added:

ASN.1 Value Notation: { lowpanObjects(1) lowpanStats(1) lowpanOutTransmits(29) }

Dotted Decimal Notation: .1.1.29

```
0D                                  # UNIVERSAL TAG 13
   03                               # 3 bytes, primitive
      01 01 1D                      # X.690 Clause 8.20
#      1  1 29                        show component encoding
```

Figure 3: MIB relative object identifier, in BER

```
D8 6E                               # tag(110)
   43                               # 0b010_01001: mt 2 (bstr), 3 bytes
      01 01 1D                      # X.690 Clause 8.20
```

Figure 4: MIB relative object identifier, in CBOR

This relative OID saves seven bytes compared to the full OID encoding.

4. Tag Factoring with Arrays and Maps

OID tags can tag byte strings (as discussed above), but also CBOR arrays and maps. The idea in the latter case is that the tag is factored out from each individual item in the container; the tag is placed on the array or map instead.

When an OID tag is applied to an array, it means that the respective tag is imputed to all elements of the array that are byte strings, arrays, or maps. (There is no effect on other elements, including text strings or tags.) For example, when an array is tagged with TBD111, every array element that is a byte string is an OID, and every element that is an array or map is in turn treated as discussed here.

When an OID tag is applied to a map, it means that the respective tag is imputed to all keys in the map that are byte strings, arrays, or maps; again, there is no effect on keys of other major types. Note that there is also no effect on the values in the map.

As a result of these rules, tag factoring in nested arrays and maps is supported. For example, a 3-dimensional array of OIDs can be composed by using a single TBD111 tag containing an array of arrays of arrays of byte strings. All such byte strings are then considered OIDs.

4.1. Tag Factoring Example: X.500 Distinguished Name

Consider the X.500 distinguished name:

Attribute Types	Attribute Values
c (2.5.4.6)	US
l (2.5.4.7) s (2.5.4.8) postalCode (2.5.4.17)	Los Angeles CA 90013
street (2.5.4.9)	532 S Olive St
businessCategory (2.5.4.15) buildingName (0.9.2342.19200300.100.1.48)	Public Park Pershing Square

Table 1: Example X.500 Distinguished Name

Table 1 has four "relative distinguished names" (RDNs). The country and street RDNs are single-valued. The second and fourth RDNs are multi-valued.

The equivalent representations in CBOR diagnostic notation (Section 8 of [RFC8949]) and CBOR are:

```
111([{ h'550406': "US" },
     { h'550407': "Los Angeles", h'550408': "CA",
       h'550411': "90013" },
     { h'550409': "532 S Olive St" },
     { h'55040f': "Public Park",
       h'0992268993f22c640130': "Pershing Square" }])
```

Figure 5: Distinguished Name, in CBOR Diagnostic Notation

```
d8 6f                                   # tag(111)
   84                                   # array(4)
      a1                                # map(1)
         43 550406                      # 2.5.4.6 (4)
         62                             # text(2)
            5553                        # "US"
      a3                                # map(3)
         43 550407                      # 2.5.4.7 (4)
         6b                             # text(11)
            4c6f7320416e67656c6573      # "Los Angeles"
         43 550408                      # 2.5.4.8 (4)
         62                             # text(2)
            4341                        # "CA"
         43 550411                      # 2.5.4.17 (4)
         65                             # text(5)
            3930303133                  # "90013"
      a1                                # map(1)
         43 550409                      # 2.5.4.9 (4)
         6e                             # text(14)
            3533322053204f6c697665205374   # "532 S Olive St"
      a2                                # map(2)
         43 55040f                      # 2.5.4.15 (4)
         6b                             # text(11)
            5075626c6963205061726b      # "Public Park"
         4a 0992268993f22c640130    # 0.9.2342.19200300.100.1.48 (11)
         6f                             # text(15)
            5065727368696e6720537175617265 # "Pershing Square"
```

Figure 6: Distinguished Name, in CBOR (109 bytes)

(This example encoding assumes that all attribute values are UTF-8 strings, or can be represented as UTF-8 strings with no loss of information.)

5. CDDL Control Operators

CDDL specifications may want to specify the use of SDNVs or SDNV sequences (as defined for the tag content for TBD110). This document introduces two new control operators that can be applied to a target value that is a byte string:

* ".sdnv", with a control type that contains unsigned integers. The byte string is specified to be encoded as an [RFC6256] SDNV (BER encoding) for the matching values of the control type.

* ".sdnvseq", with a control type that contains arrays of unsigned integers. The byte string is specified to be encoded as a sequence of [RFC6256] SDNVs (BER encoding) that decodes to an array of unsigned integers matching the control type.

* ".oid", like ".sdnvseq", except that the X*40+Y translation for absolute OIDs is included (see Figure 8).

Figure 7 shows an example for the use of ".sdnvseq" for a part of a structure using OIDs that could be used in Figure 6; Figure 8 shows the same with the ".oid" operator.

```
country-rdn = {country-oid => country-value}
country-oid = bytes .sdnvseq [85, 4, 6]
country-value = text .size 2
```

Figure 7: Using .sdnvseq

```
country-rdn = {country-oid => country-value}
country-oid = bytes .oid [2, 5, 4, 6]
country-value = text .size 2
```

Figure 8: Using .oid

Note that the control type need not be a literal; e.g., "bytes .oid [2, 5, 4, *uint]" matches all OIDs inside OID arc 2.5.4, "attributeType".

6. CDDL typenames

For the use with CDDL [RFC8610], the typenames defined in Figure 9 are recommended:

```
oid = #6.111(bstr)
roid = #6.110(bstr)
pen = #6.112(bstr)
```

Figure 9: Recommended typenames for CDDL

7. IANA Considerations

7.1. CBOR Tags

IANA is requested to assign the CBOR tags in Table 2, with the present document as the specification reference.

Tag	Data Item	Semantics
TBD111	byte string or array or map	object identifier (BER encoding)
TBD110	byte string or array or map	relative object identifier (BER encoding); SDNV [RFC6256] sequence
TBD112	byte string or array or map	object identifier (BER encoding), relative to 1.3.6.1.4.1

Table 2: Values for New Tags

7.2. CDDL Control Operators

IANA is requested to assign the CDDL Control Operators in Table 3, with the present document as the specification reference.

Name	Reference
.sdnv	[this document, Section 5]
.sdnvseq	[this document, Section 5]
.oid	[this document, Section 5]

Table 3: New CDDL Operators

8. Security Considerations

The security considerations of [RFC8949] apply.

The encodings in Clauses 8.19 and 8.20 of [X.690] are quite compact and unambiguous, but MUST be followed precisely to avoid security pitfalls. In particular, the requirements set out in Section 2.1 of this document need to be followed; otherwise, an attacker may be able to subvert a checking process by submitting alternative representations that are later taken as the original (or even something else entirely) by another decoder supposed to be protected by the checking process.

OIDs and relative OIDs can always be treated as opaque byte strings. Actually understanding the structure that was used for generating them is not necessary, and, except for checking the structure requirements, it is strongly NOT RECOMMENDED to perform any processing of this kind (e.g., converting into dotted notation and back) unless absolutely necessary. If the OIDs are translated into other representations, the usual security considerations for non-trivial representation conversions apply; the integer values are unlimited in range.

9. References

9.1. Normative References

[RFC2119] 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>.

[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>.

[RFC8174] 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>.

[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>.

[RFC8949] 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>.

[X.660] International Telecommunications Union, "Information technology Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree", ITU-T Recommendation X.660, July 2011, <https://www.itu.int/rec/T-REC-X.660>.

[X.680] International Telecommunications Union, "Information technology Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, August 2015, <https://www.itu.int/rec/T-REC-X.680>.

[X.690] International Telecommunications Union, "Information technology ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015, <https://www.itu.int/rec/T-REC-X.690>.

9.2. Informative References

[IANA.enterprise-numbers] IANA, "enterprise-numbers", <http://www.iana.org/assignments/enterprise-numbers>.

[OID-INFO] Orange SA, "OID Repository", 2016, <http://www.oid-info.com/>.

[PCRE] Ho, A., "PCRE - Perl Compatible Regular Expressions", 2018, <http://www.pcre.org/>.

[RFC7388] Schoenwaelder, J., Sehgal, A., Tsou, T., and C. Zhou, "Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7388, DOI 10.17487/RFC7388, October 2014, <https://www.rfc-editor.org/info/rfc7388>.

[X.672] International Telecommunications Union, "Information technology Open systems interconnection Object identifier resolution system", ITU-T Recommendation X.672, August 2010, <https://www.itu.int/rec/T-REC-X.672>.

Appendix A. Change Log

This section is to be removed before publishing as an RFC.

A.1. Changes from -04 to -05

* Update acknowledgements, contributor list, and author list

A.2. Changes from -03 to -04

Process WGLC and shepherd comments:

* Update references (RFC 8949, URIs for ITU-T)

* Define arc for this document, reference SDN definition

* Restructure, small editorial clarifications

A.3. Changes from -02 to -03

* Add tag TBD112 for PEN-relative OIDs

* Add suggested CDDL typenames; reference RFC8610

A.4. Changes from -01 to -02

Minor editorial changes, remove some remnants, ready for WGLC.

A.5. Changes from -00 to -01

Clean up OID tag factoring.

A.6. Changes from -07 (bormann) to -00 (ietf)

Resubmitted as WG draft after adoption.

A.7. Changes from -06 to -07

Reduce the draft back to its basic mandate: Describe CBOR tags for what is colloquially know as ASN.1 Object IDs.

A.8. Changes from -05 to -06

Refreshed the draft to the current date ("keep-alive").

A.9. Changes from -04 to -05

Discussed UUID usage in CBOR, and incorporated fixes proposed by Olivier Dubuisson, including fixes regarding OID nomenclature.

A.10. Changes from -03 to -04

Changes occurred based on limited feedback, mainly centered around the abstract and introduction, rather than substantive technical changes. These changes include:

* Changed the title so that it is about tags and techniques.

* Rewrote the abstract to describe the content more accurately, and to point out that no changes to the wire protocol are being proposed.

* Removed "ASN.1" from "object identifiers", as OIDs are independent of ASN.1.

* Rewrote the introduction to be more about the present text.

* Proposed a concise OID arc.

* Provided binary regular expression forms for OID validation.

* Updated IANA registration tables.

A.11. Changes from -02 to -03

Many significant changes occurred in this version. These changes include:

* Expanded the draft scope to be a comprehensive CBOR update.

* Added OID-related sections: OID Enumerations, OID Maps and Arrays, and Applications and Examples of OIDs.

* Added Tag 36 update (binary MIME, better definitions).

* Added stub/experimental sections for X.690 Series Tags (tag <<X>>) and Regular Expressions (tag 35).

* Added technique for representing sets and multisets.

* Added references and fixed typos.

Acknowledgments

Sean Leonard started the work on this document in 2014 with an elaborate proposal. Jim Schaad provided a significant review of this document.

Contributors

Sean Leonard
Penango, Inc.
5900 Wilshire Boulevard
21st Floor
Los Angeles, CA, 90036
United States of America

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

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 1 October 2021

C. Bormann
Universität Bremen TZI

30 March 2021

Concise Binary Object Representation (CBOR) Tags for Object Identifiers
draft-ietf-cbor-tags-oid-06

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

The present document defines CBOR tags for object identifiers (OIDs). It is intended as the reference document for the IANA registration of the CBOR tags so defined.

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 1 October 2021.

Copyright Notice

Table of Contents

1. Introduction

The Concise Binary Object Representation (CBOR, [RFC8949]) provides for the interchange of structured data without a requirement for a pre-agreed schema. [RFC8949] defines a basic set of data types, as well as a tagging mechanism that enables extending the set of data types supported via an IANA registry.

The present document defines CBOR tags for object identifiers (OIDs, [X.660]), which many IETF protocols carry. The ASN.1 Basic Encoding Rules (BER, [X.690]) specify binary encodings of both (absolute) object identifiers and relative object identifiers. The contents of these encodings (the "value" part of BER's type-length-value structure) can be carried in a CBOR byte string. This document defines two CBOR tags that cover the two kinds of ASN.1 object identifiers encoded in this way. The tags can also be applied to arrays and maps to efficiently tag all elements of an array or all keys of a map. It is intended as the reference document for the IANA registration of the tags so defined.

1.1. Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The terminology of [RFC8949] applies; in particular the term "byte" is used in its now customary sense as a synonym for "octet". The term "SDNV" is used as defined in [RFC6256].

2. Object Identifiers

The International Object Identifier tree [X.660] is a hierarchically managed space of identifiers, each of which is uniquely represented as a sequence of unsigned integer values [X.680]. (These integer values are called "primary integer values" in X.660 because they can be accompanied by (not necessarily unambiguous) secondary identifiers. We ignore the latter and simply use the term "integer values" here, occasionally calling out their unsignedness. We also use the term "arc" when the focus is on the edge of the tree labeled by such an integer value, as well as in the sense of a "long arc", i.e. a (sub)sequence of such integer values.)

While these sequences can easily be represented in CBOR arrays of unsigned integers, a more compact representation can often be achieved by adopting the widely used representation of object identifiers defined in BER; this representation may also be more amenable to processing by other software that makes use of object identifiers.

BER represents the sequence of unsigned integers by concatenating self-delimiting [RFC6256] representations of each of the integer values in sequence.

ASN.1 distinguishes absolute object identifiers (ASN.1 Type "OBJECT IDENTIFIER"), which begin at a root arc ([X.660] Clause 3.5.21), from relative object identifiers (ASN.1 Type "RELATIVE-OID"), which begin relative to some object identifier known from context ([X.680] Clause 3.8.63). As a special optimization, BER combines the first two integers in an absolute object identifier into one numeric identifier by making use of the property of the hierarchy that the first arc has only three integer values (0, 1, and 2), and the second arcs under 0 and 1 are limited to the integer values between 0 and 39. (The root arc "joint-iso-itu-t(2)" has no such limitations on its second arc.) If X and Y are the first two integer values, the single integer value actually encoded is computed as:

X * 40 + Y

The inverse transformation (again making use of the known ranges of X and Y) is applied when decoding the object identifier.

Since the semantics of absolute and relative object identifiers differ, this specification defines two tags, collectively called the "OID tags" here:

Tag TBD111: tags a byte string as the [X.690] encoding of an absolute object identifier (simply "object identifier" or "OID").

Tag TBD110: tags a byte string as the [X.690] encoding of a relative object identifier (also "relative OID"). Since the encoding of each number is the same as for [RFC6256] Self-Delimiting Numeric Values (SDNVs), this tag can also be used for tagging a byte string that contains a sequence of zero or more SDNVs.

Tag TBD112: structurally like TBD110, but understood to be relative to "1.3.6.1.4.1" (IANA Private Enterprise Number OID, [IANA.enterprise-numbers]). Hence, the semantics of the result are that of an absolute object identifier.

2.1. Requirements on the byte string being tagged

To form a valid tag, a byte string tagged by TBD111, TBD110, or TBD112 MUST be syntactically valid contents (the value part) for a BER representation of an object identifier (Sections 8.19, 8.20, and 8.20 of [X.690], respectively): A concatenation of zero or more SDNV values, where each SDNV value is a sequence of one or more bytes that all have their most significant bit set, except for the last byte, where it is unset. Also, the first byte of each SDNV cannot be a leading zero in SDNV's base-128 arithmetic, so it cannot take the value 0x80 (bullet (c) in Section 8.1.2.4.2 of [X.690]).

In other words:

* the byte string's first byte, and any byte that follows a byte that has the most significant bit unset, MUST NOT be 0x80 (this requirement requires expressing the integer values in their shortest form, with no leading zeroes)

* the byte string's last byte MUST NOT have the most significant bit set (this requirement excludes an incomplete final integer value)

If either of these invalid conditions are encountered, the tag is invalid.

[X.680] restricts RELATIVE-OID values to have at least one arc, i.e., their encoding would have at least one SDNV. This specification permits empty relative object identifiers; they may still be excluded by application semantics.

To facilitate the search for specific object ID values, it is RECOMMENDED that definite length encoding (see Section 3.2.3 of [RFC8949]) is used for the byte strings used as tag content for these tags.

The valid set of byte strings can also be expressed using regular expressions on bytes, using no specific notation but resembling [PCRE]. Unlike typical regular expressions that operate on character sequences, the following regular expressions take bytes as their domain, so they can be applied directly to CBOR byte strings.

For byte strings with tag TBD111:

```
"/^(([\x81-\xFF][\x80-\xFF]*)?[\x00-\x7F])+$/"
```

For byte strings with tag TBD110 or TBD112:

```
"/^(([\x81-\xFF][\x80-\xFF]*)?[\x00-\x7F])*$/"
```

A tag with tagged content that does not conform to the applicable regexp is invalid.

2.2. Discussion

Staying close to the way object identifiers are encoded in ASN.1 BER makes back-and-forth translation easy; otherwise we would choose a more efficient encoding. Object identifiers in IETF protocols are serialized in dotted decimal form or BER form, so there is an advantage in not inventing a third form. Also, expectations of the cost of encoding object identifiers are based on BER; using a different encoding might not be aligned with these expectations. If additional information about an OID is desired, lookup services such as the OID Resolution Service (ORS) [X.672] and the OID Repository [OID-INFO] are available.

3. Basic Examples

This section gives simple examples of an absolute and a relative object identifier, represented via tag number TBD111 and TBD110, respectively.

RFC editor: These and other examples assume the allocation of 111 for TBD111 and 110 for TBD110 and need to be changed if that isn't the actual allocation. Please remove this paragraph.

3.1. Encoding of the SHA-256 OID

ASN.1 Value Notation: { joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2) sha256(1) }

Dotted Decimal Notation: 2.16.840.1.101.3.4.2.1

```
06                                  # UNIVERSAL TAG 6
   09                               # 9 bytes, primitive
      60 86 48 01 65 03 04 02 01    # X.690 Clause 8.19
#      |   840  1  |  3  4  2  1     show component encoding
#   2.16         101
```

Figure 1: SHA-256 OID in BER

```
D8 6F                               # tag(111)
   49                               # 0b010_01001: mt 2, 9 bytes
      60 86 48 01 65 03 04 02 01    # X.690 Clause 8.19
```

Figure 2: SHA-256 OID in CBOR

3.2. Encoding of a MIB Relative OID

Given some OID (e.g., "lowpanMib", assumed to be "1.3.6.1.2.1.226" [RFC7388]), to which the following is added:

ASN.1 Value Notation: { lowpanObjects(1) lowpanStats(1) lowpanOutTransmits(29) }

Dotted Decimal Notation: .1.1.29

```
0D                                  # UNIVERSAL TAG 13
   03                               # 3 bytes, primitive
      01 01 1D                      # X.690 Clause 8.20
#      1  1 29                        show component encoding
```

Figure 3: MIB relative object identifier, in BER

```
D8 6E                               # tag(110)
   43                               # 0b010_01001: mt 2 (bstr), 3 bytes
      01 01 1D                      # X.690 Clause 8.20
```

Figure 4: MIB relative object identifier, in CBOR

This relative OID saves seven bytes compared to the full OID encoding.

4. Tag Factoring with Arrays and Maps

OID tags can tag byte strings (as discussed above), but also CBOR arrays and maps. The idea in the latter case is that the tag is factored out from each individual item in the container; the tag is placed on the array or map instead.

When an OID tag is applied to an array, it means that the respective tag is imputed to all elements of the array that are byte strings, arrays, or maps. (There is no effect on other elements, including text strings or tags.) For example, when an array is tagged with TBD111, every array element that is a byte string is an OID, and every element that is an array or map is in turn treated as discussed here.

When an OID tag is applied to a map, it means that the respective tag is imputed to all keys in the map that are byte strings, arrays, or maps; again, there is no effect on keys of other major types. Note that there is also no effect on the values in the map.

As a result of these rules, tag factoring in nested arrays and maps is supported. For example, a 3-dimensional array of OIDs can be composed by using a single TBD111 tag containing an array of arrays of arrays of byte strings. All such byte strings are then considered OIDs.

4.1. Tag Factoring Example: X.500 Distinguished Name

Consider the X.500 distinguished name:

Attribute Types	Attribute Values
c (2.5.4.6)	US
l (2.5.4.7) s (2.5.4.8) postalCode (2.5.4.17)	Los Angeles CA 90013
street (2.5.4.9)	532 S Olive St
businessCategory (2.5.4.15) buildingName (0.9.2342.19200300.100.1.48)	Public Park Pershing Square

Table 1: Example X.500 Distinguished Name

Table 1 has four "relative distinguished names" (RDNs). The country and street RDNs are single-valued. The second and fourth RDNs are multi-valued.

The equivalent representations in CBOR diagnostic notation (Section 8 of [RFC8949]) and CBOR are:

```
111([{ h'550406': "US" },
     { h'550407': "Los Angeles", h'550408': "CA",
       h'550411': "90013" },
     { h'550409': "532 S Olive St" },
     { h'55040f': "Public Park",
       h'0992268993f22c640130': "Pershing Square" }])
```

Figure 5: Distinguished Name, in CBOR Diagnostic Notation

```
d8 6f                                   # tag(111)
   84                                   # array(4)
      a1                                # map(1)
         43 550406                      # 2.5.4.6 (4)
         62                             # text(2)
            5553                        # "US"
      a3                                # map(3)
         43 550407                      # 2.5.4.7 (4)
         6b                             # text(11)
            4c6f7320416e67656c6573      # "Los Angeles"
         43 550408                      # 2.5.4.8 (4)
         62                             # text(2)
            4341                        # "CA"
         43 550411                      # 2.5.4.17 (4)
         65                             # text(5)
            3930303133                  # "90013"
      a1                                # map(1)
         43 550409                      # 2.5.4.9 (4)
         6e                             # text(14)
            3533322053204f6c697665205374   # "532 S Olive St"
      a2                                # map(2)
         43 55040f                      # 2.5.4.15 (4)
         6b                             # text(11)
            5075626c6963205061726b      # "Public Park"
         4a 0992268993f22c640130    # 0.9.2342.19200300.100.1.48 (11)
         6f                             # text(15)
            5065727368696e6720537175617265 # "Pershing Square"
```

Figure 6: Distinguished Name, in CBOR (109 bytes)

(This example encoding assumes that all attribute values are UTF-8 strings, or can be represented as UTF-8 strings with no loss of information.)

5. CDDL Control Operators

CDDL specifications may want to specify the use of SDNVs or SDNV sequences (as defined for the tag content for TBD110). This document introduces two new control operators that can be applied to a target value that is a byte string:

* ".sdnv", with a control type that contains unsigned integers. The byte string is specified to be encoded as an [RFC6256] SDNV (BER encoding) for the matching values of the control type.

* ".sdnvseq", with a control type that contains arrays of unsigned integers. The byte string is specified to be encoded as a sequence of [RFC6256] SDNVs (BER encoding) that decodes to an array of unsigned integers matching the control type.

* ".oid", like ".sdnvseq", except that the X*40+Y translation for absolute OIDs is included (see Figure 8).

Figure 7 shows an example for the use of ".sdnvseq" for a part of a structure using OIDs that could be used in Figure 6; Figure 8 shows the same with the ".oid" operator.

```
country-rdn = {country-oid => country-value}
country-oid = bytes .sdnvseq [85, 4, 6]
country-value = text .size 2
```

Figure 7: Using .sdnvseq

```
country-rdn = {country-oid => country-value}
country-oid = bytes .oid [2, 5, 4, 6]
country-value = text .size 2
```

Figure 8: Using .oid

Note that the control type need not be a literal; e.g., "bytes .oid [2, 5, 4, *uint]" matches all OIDs inside OID arc 2.5.4, "attributeType".

6. CDDL typenames

For the use with CDDL [RFC8610], the typenames defined in Figure 9 are recommended:

```
oid = #6.111(bstr)
roid = #6.110(bstr)
pen = #6.112(bstr)
```

Figure 9: Recommended typenames for CDDL

7. IANA Considerations

7.1. CBOR Tags

IANA is requested to assign the CBOR tags in Table 2, with the present document as the specification reference.

Tag	Data Item	Semantics
TBD111	byte string or array or map	object identifier (BER encoding)
TBD110	byte string or array or map	relative object identifier (BER encoding); SDNV [RFC6256] sequence
TBD112	byte string or array or map	object identifier (BER encoding), relative to 1.3.6.1.4.1

Table 2: Values for New Tags

7.2. CDDL Control Operators

IANA is requested to assign the CDDL Control Operators in Table 3, with the present document as the specification reference.

Name	Reference
.sdnv	[this document, Section 5]
.sdnvseq	[this document, Section 5]
.oid	[this document, Section 5]

Table 3: New CDDL Operators

8. Security Considerations

The security considerations of [RFC8949] apply.

The encodings in Clauses 8.19 and 8.20 of [X.690] are quite compact and unambiguous, but MUST be followed precisely to avoid security pitfalls. In particular, the requirements set out in Section 2.1 of this document need to be followed; otherwise, an attacker may be able to subvert a checking process by submitting alternative representations that are later taken as the original (or even something else entirely) by another decoder supposed to be protected by the checking process.

OIDs and relative OIDs can always be treated as opaque byte strings. Actually understanding the structure that was used for generating them is not necessary, and, except for checking the structure requirements, it is strongly NOT RECOMMENDED to perform any processing of this kind (e.g., converting into dotted notation and back) unless absolutely necessary. If the OIDs are translated into other representations, the usual security considerations for non-trivial representation conversions apply; the integer values are unlimited in range.

9. References

9.1. Normative References

[RFC2119] 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>.

[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>.

[RFC8174] 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>.

[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>.

[RFC8949] 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>.

[X.660] International Telecommunications Union, "Information technology Procedures for the operation of object identifier registration authorities: General procedures and top arcs of the international object identifier tree", ITU-T Recommendation X.660, July 2011, <https://www.itu.int/rec/T-REC-X.660>.

[X.680] International Telecommunications Union, "Information technology Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, August 2015, <https://www.itu.int/rec/T-REC-X.680>.

[X.690] International Telecommunications Union, "Information technology ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015, <https://www.itu.int/rec/T-REC-X.690>.

9.2. Informative References

[IANA.enterprise-numbers] IANA, "enterprise-numbers", <http://www.iana.org/assignments/enterprise-numbers>.

[OID-INFO] Orange SA, "OID Repository", 2016, <http://www.oid-info.com/>.

[PCRE] Ho, A., "PCRE - Perl Compatible Regular Expressions", 2018, <http://www.pcre.org/>.

[RFC7388] Schoenwaelder, J., Sehgal, A., Tsou, T., and C. Zhou, "Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7388, DOI 10.17487/RFC7388, October 2014, <https://www.rfc-editor.org/info/rfc7388>.

[X.672] International Telecommunications Union, "Information technology Open systems interconnection Object identifier resolution system", ITU-T Recommendation X.672, August 2010, <https://www.itu.int/rec/T-REC-X.672>.

Appendix A. Change Log

This section is to be removed before publishing as an RFC.

A.1. Changes from -05 to -06

Add references to specific section numbers of [X.690] to better explain validity of enclosed byte string.

A.2. Changes from -04 to -05

* Update acknowledgements, contributor list, and author list

A.3. Changes from -03 to -04

Process WGLC and shepherd comments:

* Update references (RFC 8949, URIs for ITU-T)

* Define arc for this document, reference SDN definition

* Restructure, small editorial clarifications

A.4. Changes from -02 to -03

* Add tag TBD112 for PEN-relative OIDs

* Add suggested CDDL typenames; reference RFC8610

A.5. Changes from -01 to -02

Minor editorial changes, remove some remnants, ready for WGLC.

A.6. Changes from -00 to -01

Clean up OID tag factoring.

A.7. Changes from -07 (bormann) to -00 (ietf)

Resubmitted as WG draft after adoption.

A.8. Changes from -06 to -07

Reduce the draft back to its basic mandate: Describe CBOR tags for what is colloquially know as ASN.1 Object IDs.

A.9. Changes from -05 to -06

Refreshed the draft to the current date ("keep-alive").

A.10. Changes from -04 to -05

Discussed UUID usage in CBOR, and incorporated fixes proposed by Olivier Dubuisson, including fixes regarding OID nomenclature.

A.11. Changes from -03 to -04

Changes occurred based on limited feedback, mainly centered around the abstract and introduction, rather than substantive technical changes. These changes include:

* Changed the title so that it is about tags and techniques.

* Rewrote the abstract to describe the content more accurately, and to point out that no changes to the wire protocol are being proposed.

* Removed "ASN.1" from "object identifiers", as OIDs are independent of ASN.1.

* Rewrote the introduction to be more about the present text.

* Proposed a concise OID arc.

* Provided binary regular expression forms for OID validation.

* Updated IANA registration tables.

A.12. Changes from -02 to -03

Many significant changes occurred in this version. These changes include:

* Expanded the draft scope to be a comprehensive CBOR update.

* Added OID-related sections: OID Enumerations, OID Maps and Arrays, and Applications and Examples of OIDs.

* Added Tag 36 update (binary MIME, better definitions).

* Added stub/experimental sections for X.690 Series Tags (tag <<X>>) and Regular Expressions (tag 35).

* Added technique for representing sets and multisets.

* Added references and fixed typos.

Acknowledgments

Sean Leonard started the work on this document in 2014 with an elaborate proposal. Jim Schaad provided a significant review of this document.

Contributors

Sean Leonard
Penango, Inc.
5900 Wilshire Boulevard
21st Floor

Los Angeles, CA, 90036
United States of America

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

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

anima Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 July 2021

M. Richardson
Sandelman Software Works

21 January 2021

On storing CBOR encoded items on stable storage
draft-richardson-cbor-file-magic-01

Abstract

This document proposes an on-disk format for CBOR objects that is friendly to common on-disk recognition systems like the Unix file(1) command.

This document is being discussed at: https://github.com/mcr/cbor-magic-number

Table of Contents

1. Introduction

Since very early in computing, operating systems have sought ways to mark which files could be processed by which programs.

For instance, the Unix file(1) command, which has existed since 1973 ([file]), has been able to identify many file formats for decades. Many systems (Linux, MacOS, Windows) will select the correct application based upon the file contents, if the system can not determine it by other means: for instsance, MacOS maintains a resource fork that includes MIME information and therefore ideally never needs to know what anything about the file. Other systems do this by file extensions.

While having a MIME type associated with the file is a better solution in general, when files become disconnected from their type information, such as when attempting to do forensics on a damaged system, then being able to identify a file type can become very important.

It is noted that in the MIME type registration, that a magic number is asked for, if available, as is a file extension.

A challenge for the file(1) program is often that it can be confused by the encoding vs the content. For instance, an Android "apk" used to transfer and store an application may be identified as a ZIP file. Both OpenOffice or MSOffice files are XML files, but appear as ZIP, unless they are flat files, in which case they appear to be generic XML files.

As CBOR becomes a more and more common encoding for a wide variety of artifacts, identifying them as CBOR is probably not useful. This document provides a way to encode a magic number into the beginning of a CBOR format file. Two options are presented, with the intention of standardizing only one.

These proposals are invasive to how CBOR protocols are written to disk, but in both cases, the proposed envelope does not require that the tag be transfered on the wire.

In addition to the on-disk identification aspects, there are some protocols which may benefit from having such a magic on the wire if they presently using a different (legacy) encoding scheme. The presence of the identifiable magic sequence signals that CBOR is being used or a legacy scheme.

2. Requirements for a Magic Number

A magic number is ideally a unique fingerprint, present in the first 4 or 8 bytes of the file, which does not change when the content change, and does not depend upon the length of the file.

Less ideal solutions have a pattern that needs to be matched, but in which some bytes need to be ignored. While the Unix file(1) command can be told to ignore bytes, this can lead to ambiguities.

3. Protocol Proposal

This proposal makes use of CBOR Sequences as described in [RFC8742].

This proposal consists of two tags and a constant string for a total of 12 bytes.

1. The file shall start with the Self-described CBOR tag, 55799, as described in [RFC8949] section 3.4.6.

2. The file shall continue with a CBOR tag, from the First Come First Served space, which uniquely identifies the CBOR Protocol. The use of a four-byte tag is encouraged.

3. The three byte CBOR array containing 0x42_4F_52. When encoded it shows up as "CBOR"

The first part identifies the file as being CBOR, and does so with all the desirable properties explained in Specifically, it does not seem to conflict with any known file types, and it is not valid Unicode.[RFC8949] section 3.4.6.

The second part identifies which CBOR Protocol is used. CBOR Protocol designers should obtain a tag for each major object that they might store on disk. As there are more than 4 million available 4-byte tags, there should be issue in allocating a few to all available CBOR Protocols. The policy is First Come First Served, so all that is required is an email to IANA, having filled in the small template provided in section 9.2 of [RFC8949].

The third part is a constant value 0x43_42_4f_52, "CBOR". This means that should a file be reviewed by a human (directly in an editor, or in a hexdump display), it will include the string "CBOR" prominently. The value is also included because the two tags need to tag something.

4. Security Considerations

This document provides a way to identify CBOR Protocol objects. Clearly identifying CBOR contents on disk may have a variety of impacts.

The most obvious is that it may allow malware to identify interesting objects on disk, and then corrupt them.

5. IANA Considerations

This document makes no new requests to IANA.

6. Acknowledgements

The CBOR WG brainstormed this protocol on January 20, 2021.

7. Changelog

8. References

8.1. Normative References

[BCP14] 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>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC8949] 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>.

8.2. Informative References

[file] Wikipedia, "file (command)", 20 January 2021, <https://en.wikipedia.org/wiki/File_%28command%29>.

[ilbm] Wikipedia, "Interleaved BitMap", 20 January 2021, <https://en.wikipedia.org/wiki/ILBM>.

Contributors

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

anima Working Group
Internet-Draft
Intended status: Standards Track
Expires: 10 August 2021

M. Richardson
Sandelman Software Works

6 February 2021

CBOR tags for IPv4 and IPv6 addresses and prefixes
draft-richardson-cbor-network-addresses-01

Abstract

This document describes two CBOR Tags to be used with IPv4 and IPv6 addresses and prefixes.

RFC-EDITOR-please remove: This work is tracked at https://github.com/mcr/cbor-network-address.git

Table of Contents

1. Introduction

[RFC8949] defines a number of CBOR Tags for common items.

Not included are ones to indicate if the item is an IPv4 or IPv6 address, or if it is an address plus prefix length. This document defines them.

2. Protocol

These tags can applied to byte strings to represent a single address.

When applied to an array, the represent a CIDR-style prefix. When a byte string (without prefix) appears in a context where a prefix is expected, then it is to be assumed that all bits are relevant. That is, for IPv4, a /32 is implied, and for IPv6, a /128 is implied.

2.1. IPv6

IANA has allocated tag TBD1 for IPv6 uses.

An IPv6 address is to be encoded as up to sixteen-byte bytestring ([RFC8949] section, 3.1, major type 2), prefixed with tag TBD1. Trailing zero octets may be omitted.

An IPv6 prefix, such as 2001:db8:1234::/48 is to be encoded as a two element array, with the length of the prefix first:

```
TBD1([ 48, h'20010db81234'])
```

2.2. IPv4

IANA has allocated tag TBD2 for IPv4 uses.

An IPv4 address is to be encoded as a four-byte bytestring ([RFC8949] section, 3.1, major type 2), prefixed with tag TBD2. Trailing zero octets may be omitted.

An IPv4 prefix, such as 192.0.2.1/24 is to be encoded as a two element array, with the length of the prefix first:

```
TBD2([ 24, h'C0000201'])
```

3. Security Considerations

Identifying which byte sequences in a protocol are addresses may allow an attacker or eavesdropper to better understand what parts of a packet to attack.

Reading the relevant RFC may provide more information, so it would seem that any additional security that was provided by not being able to identify what are IP addresses falls into the security by obscurity category.

4. IANA Considerations

IANA is asked to allocate two tags from the Specification Required area of the Concise Binary Object Representation (CBOR) Tags, in the ("1+1") area.

4.1. TBD1 - IPv6

Data Item: byte-string and array
Semantics: IPv6 or [IPv6,prefixlen]

4.2. TBD2 - IPv4

Data Item: byte-string and array
Semantics: IPv4 or [IPv4,prefixlen]

5. Acknowledgements

none yet

6. Changelog

7. Normative References

[BCP14] 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>.

[RFC8949] 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>.

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca