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.

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.

Copyright Notice

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

\[ \text{dt'2019-07-21T19:53Z'} \]

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:

```cddl
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:

```cddl
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-z0-9@\$](([-.])*[A-Za-z0-9@\$])"*
op = ".." / "..." /
text .regexp \"\[A-Za-z0-9@\$](([-.])*[A-Za-z0-9@\$])\"
namea = ["name", id] / ["gen", id, +type]
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
10.2. Informative References


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

[I-D.ietf-core-coral]


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
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.

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 27 December 2021.

Copyright Notice

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

See also [I-D.draft-bormann-jsonpath-iregexp], which could be specifically called out via ".iregexp" (even though ".regexp" as per Section 3.8.3 of [RFC8610] would also have the same semantics).

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:

```plaintext
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:

```plaintext
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.

6.2. Cross-universe references

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

6.2.1. IANA references

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

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

An example of such a usage might be:

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

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

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

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@_$]{0,999}"* 
op = "." / ".." / "..." / 
text .regexp "\.[A-Za-z@_$]{0,999}"* 
namea = ["name", id] / ["gen", id, +type]
type = value / namea / ["op", op, type, type] / 
["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,
10.2. Informative References


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
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.

Notable CBOR Tags
draft-bormann-cbor-notable-tags-03
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 16 August 2021.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 3
   1.1. Terminology ........................................ 3
2. RFC 7049 (CBOR) ........................................... 3
   2.1. Tags Related to Those Defined in RFC 7049 .......... 5
3. Security .................................................. 5
   3.1. RFC 8152 (COSE) .................................... 5
   3.2. RFC 8392 (CWT) .................................... 6
4. CBOR-based Representation Formats .......................... 6
   4.1. YANG-CBOR ........................................ 7
5. Protocols ................................................ 7
   5.1. DOTS ............................................... 7
   5.2. RAINS ............................................. 8
6. Datatypes ................................................ 8
   6.1. Advanced arithmetic ................................ 8
   6.2. Variants of undefined .............................. 8
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.
<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
<th>Section of RFC 7049</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UTF-8 string</td>
<td>Standard date/time string</td>
<td>2.4.1</td>
</tr>
<tr>
<td>1</td>
<td>multiple</td>
<td>Epoch-based date/time</td>
<td>2.4.1</td>
</tr>
<tr>
<td>2</td>
<td>byte string</td>
<td>Positive bignum</td>
<td>2.4.2</td>
</tr>
<tr>
<td>3</td>
<td>byte string</td>
<td>Negative bignum</td>
<td>2.4.2</td>
</tr>
<tr>
<td>4</td>
<td>array</td>
<td>Decimal fraction</td>
<td>2.4.3</td>
</tr>
<tr>
<td>5</td>
<td>array</td>
<td>Bigfloat</td>
<td>2.4.3</td>
</tr>
<tr>
<td>21</td>
<td>multiple</td>
<td>Expected conversion to base64url encoding</td>
<td>2.4.4.2</td>
</tr>
<tr>
<td>22</td>
<td>multiple</td>
<td>Expected conversion to base64 encoding</td>
<td>2.4.4.2</td>
</tr>
<tr>
<td>23</td>
<td>multiple</td>
<td>Expected conversion to base16 encoding</td>
<td>2.4.4.2</td>
</tr>
<tr>
<td>24</td>
<td>byte string</td>
<td>Encoded CBOR data item</td>
<td>2.4.4.1</td>
</tr>
<tr>
<td>32</td>
<td>UTF-8 string</td>
<td>URI</td>
<td>2.4.4.3</td>
</tr>
<tr>
<td>33</td>
<td>UTF-8 string</td>
<td>base64url</td>
<td>2.4.4.3</td>
</tr>
<tr>
<td>34</td>
<td>UTF-8 string</td>
<td>base64</td>
<td>2.4.4.3</td>
</tr>
<tr>
<td>35</td>
<td>UTF-8 string</td>
<td>Regular expression</td>
<td>2.4.4.3</td>
</tr>
<tr>
<td>36</td>
<td>UTF-8 string</td>
<td>MIME message</td>
<td>2.4.4.3</td>
</tr>
<tr>
<td>55799</td>
<td>multiple</td>
<td>Self-describe CBOR</td>
<td>2.4.5</td>
</tr>
</tbody>
</table>

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 item.

* 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].
### Table 2: Tag numbers defined in RFC 8152, COSE

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>COSE_Encrypt0</td>
<td>COSE Single Recipient Encrypted Data Object</td>
</tr>
<tr>
<td>17</td>
<td>COSE_Mac0</td>
<td>COSE Mac w/o Recipients Object</td>
</tr>
<tr>
<td>18</td>
<td>COSE_Sign1</td>
<td>COSE Single Signer Data Object</td>
</tr>
<tr>
<td>96</td>
<td>COSE_Encrypt</td>
<td>COSE Encrypted Data Object</td>
</tr>
<tr>
<td>97</td>
<td>COSE_Mac</td>
<td>COSE MACed Data Object</td>
</tr>
<tr>
<td>98</td>
<td>COSE_Sign</td>
<td>COSE Signed Data Object</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>CBOR Web Token (CWT)</td>
<td>CBOR Web Token (CWT)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
<th>Section of YANG-CBOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>byte string</td>
<td>YANG bits datatype</td>
<td>6.7</td>
</tr>
<tr>
<td>44</td>
<td>unsigned integer</td>
<td>YANG enumeration datatype</td>
<td>6.6</td>
</tr>
<tr>
<td>45</td>
<td>unsigned integer or text string</td>
<td>YANG identityref datatype</td>
<td>6.10</td>
</tr>
<tr>
<td>46</td>
<td>unsigned integer or text string or array</td>
<td>YANG instance-identifier datatype</td>
<td>6.13</td>
</tr>
<tr>
<td>47</td>
<td>unsigned integer</td>
<td>YANG Schema Item</td>
<td>3.2</td>
</tr>
</tbody>
</table>

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.)

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>array</td>
<td>Rational number</td>
<td>rational.html</td>
</tr>
<tr>
<td>264</td>
<td>array</td>
<td>Decimal fraction with arbitrary exponent</td>
<td>bigfrac.html</td>
</tr>
<tr>
<td>265</td>
<td>array</td>
<td>Bigfloat with arbitrary exponent</td>
<td>bigfrac.html</td>
</tr>
<tr>
<td>268</td>
<td>array</td>
<td>Extended decimal fraction</td>
<td>extended.html</td>
</tr>
<tr>
<td>269</td>
<td>array</td>
<td>Extended bigfloat</td>
<td>extended.html</td>
</tr>
<tr>
<td>270</td>
<td>array</td>
<td>Extended rational number</td>
<td>extended.html</td>
</tr>
</tbody>
</table>

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) |
| 38   | array       | Language-tagged string |
| 257  | byte string | Binary MIME message |
| 260  | byte string | Network Address (IPv4 or IPv6 or MAC Address) |
| 261  | array       | Network Address Prefix (IPv4 or IPv6 Address + Mask Length) |
| 263  | byte string | Hexadecimal string |
| 266  | text string | Internationalized resource |
| 267  | text string | identifier reference |

[htps://github.com/lucas-clemente/cbor-specs/blob/master/uuid.md][Lucas_Clemente]
[htps://peteroupc.github.io/CBOR/langtags.html][Peter_Occil]
[htps://peteroupc.github.io/CBOR/binarymime.html][Peter_Occil]
[htps://peteroupc.github.io/CBOR/iri.html][Peter_Occil]
[htps://www.employees.org/~ravir/cbor-network.txt][Ravi_Raju]
[htps://github.com/toravir/CBOR-Tag-Specs/blob/master/networkPrefix.md][Ravi_Raju]
[htps://github.com/toravir/CBOR-Tag-Specs/blob/master/hexString.md][Ravi_Raju]
[htps://peteroupc.github.io/CBOR/iri.html][Peter_Occil]
7.1. Extended Time Formats

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

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

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.)

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

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").

<table>
<thead>
<tr>
<th>Tag Number</th>
<th>Data Item</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>272</td>
<td>byte string</td>
<td>Non-UTF-8 CESU-8 string</td>
</tr>
<tr>
<td>273</td>
<td>byte string</td>
<td>Non-UTF-8 WTF-8 string</td>
</tr>
<tr>
<td>274</td>
<td>byte string</td>
<td>Non-UTF-8 MUTF-8 string</td>
</tr>
</tbody>
</table>

Table 9: Tag numbers for UTF-8 variants

9. Application-specific

(TO DO: Obtain permission to copy the definitions here; create proper table.)
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.
<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>(none valid)</td>
<td>always invalid</td>
<td>draft-bormann-cbor-notable-tags, Section 10.1</td>
</tr>
<tr>
<td>4294967295</td>
<td>(none valid)</td>
<td>always invalid</td>
<td>draft-bormann-cbor-notable-tags, Section 10.1</td>
</tr>
<tr>
<td>18446744073709551615</td>
<td>(none valid)</td>
<td>always invalid</td>
<td>draft-bormann-cbor-notable-tags, Section 10.1</td>
</tr>
<tr>
<td>63</td>
<td>byte string</td>
<td>Encoded CBOR Sequence [RFC8742]</td>
<td>draft-bormann-cbor-notable-tags, Section 2.1</td>
</tr>
</tbody>
</table>

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]

[IANA.cbor-tags]

[RFC8152]
13.2. Informative References

[I-D.bormann-cbor-time-tag]

[I-D.ietf-cose-rfc8152bis-algs]

[I-D.ietf-cose-rfc8152bis-struct]


Internet-Draft              Notable CBOR Tags              February 2021


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
Notable CBOR Tags
draft-bormann-cbor-notable-tags-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’s original edition, RFC 7049, defined 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.
1. Introduction .................................................. 3
   1.1. Terminology .............................................. 4
2. RFC 7049 (original CBOR specification) ......................... 4
   2.1. Tags Related to Those Defined in RFC 7049 ............. 5
   2.2. Tags from RFC 7049 not listed in RFC 8949 ........... 5
3. Security .................................................................. 6
   3.1. RFC 8152 (COSE) .......................................... 6
   3.2. RFC 8392 (CWT) .......................................... 7
4. CBOR-based Representation Formats ............................... 7
   4.1. YANG-CBOR ................................................. 8
5. Protocols ................................................................ 8
   5.1. DOTS ......................................................... 8
   5.2. RAINS ....................................................... 9
6. Datatypes .................................................................. 9
   6.1. Advanced arithmetic ....................................... 9
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 "CBOR Tags" [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 ([C], including C++14’s "0bnnn" binary literals [Cplusplus20]), except that superscript notation (example for two to the power of 64: \(2^{64}\)) denotes exponentiation; in the plain text version of this document, superscript notation is rendered in paragraph text by C-incompatible surrogate notation as seen in this example. Ranges expressed using ".." are inclusive of the limits given. Type names such as "int", "bigint" or "decfrac" are taken from Appendix D of [RFC8610], the Concise Data Definition Language (CDDL).

2. RFC 7049 (original CBOR specification)

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

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

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 item.

* 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].

2.2. Tags from RFC 7049 not listed in RFC 8949

Section G.3 of [RFC8949] states:

```
| Tag 35 is not defined by this document; the registration based on the definition in RFC 7049 remains in place.
```

The reason for this exclusion is that the definition of Tag 35 in Section 2.4.4.3 of [RFC7049], leaves too much open to ensure interoperability:
Tag 35 is for regular expressions in Perl Compatible Regular Expressions (PCRE) / JavaScript syntax [ECMA262].

Not only are two partially incompatible specifications given for the semantics, JavaScript regular expressions have also developed significantly within the decade since JavaScript 5.1 (which was referenced by [RFC7049]), making it less reliable to assume that a producing application will manage to stay within that 2011 subset.

Nonetheless, the registration is in place, so it is available for applications that simply want to mark a text string as being a regular expression roughly of the PCRE/Javascript flavor families.

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].
Table 2: Tag numbers defined in RFC 8152, COSE

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>COSE_Encrypt0</td>
<td>COSE Single Recipient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted Data Object</td>
</tr>
<tr>
<td>17</td>
<td>COSE_Mac0</td>
<td>COSE Mac w/o</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recipients Object</td>
</tr>
<tr>
<td>18</td>
<td>COSE_Sign1</td>
<td>COSE Single Signer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Object</td>
</tr>
<tr>
<td>96</td>
<td>COSE_Encrypt</td>
<td>COSE Encrypted Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object</td>
</tr>
<tr>
<td>97</td>
<td>COSE_Mac</td>
<td>COSE MACed Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object</td>
</tr>
<tr>
<td>98</td>
<td>COSE_Sign</td>
<td>COSE Signed Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>CBOR Web Token (CWT)</td>
<td>CBOR Web Token (CWT)</td>
</tr>
</tbody>
</table>

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.

Table 2: Tag numbers defined in RFC 8151, COSE

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>COSE_Encrypt0</td>
<td>COSE Single Recipient</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encrypted Data Object</td>
</tr>
<tr>
<td>17</td>
<td>COSE_Mac0</td>
<td>COSE Mac w/o</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recipients Object</td>
</tr>
<tr>
<td>18</td>
<td>COSE_Sign1</td>
<td>COSE Single Signer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data Object</td>
</tr>
<tr>
<td>96</td>
<td>COSE_Encrypt</td>
<td>COSE Encrypted Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object</td>
</tr>
<tr>
<td>97</td>
<td>COSE_Mac</td>
<td>COSE MACed Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object</td>
</tr>
<tr>
<td>98</td>
<td>COSE_Sign</td>
<td>COSE Signed Data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Object</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>CBOR Web Token (CWT)</td>
<td>CBOR Web Token (CWT)</td>
</tr>
</tbody>
</table>

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.

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

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]. (The seemingly random tag number was chosen so that, when represented as an encoded CBOR tag argument, it contains the Unicode character "" (U+96E8) in UTF-8, which represents rain in a number of languages.)

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.

<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>array</td>
<td>Rational number</td>
<td>rational.html</td>
</tr>
<tr>
<td>264</td>
<td>array</td>
<td>Decimal fraction with arbitrary exponent</td>
<td>bigfrac.html</td>
</tr>
<tr>
<td>265</td>
<td>array</td>
<td>Bigfloat with arbitrary exponent</td>
<td>bigfrac.html</td>
</tr>
<tr>
<td>268</td>
<td>array</td>
<td>Extended decimal fraction</td>
<td>extended.html</td>
</tr>
<tr>
<td>269</td>
<td>array</td>
<td>Extended bigfloat</td>
<td>extended.html</td>
</tr>
<tr>
<td>270</td>
<td>array</td>
<td>Extended rational number</td>
<td>extended.html</td>
</tr>
</tbody>
</table>

Table 5: Tags for advanced arithmetic
CBOR's basic generic data model (Section 2 of [RFC8949]) has a number system with limited-range integers (major types 0 and 1: \(-2^{64}..2^{64}-1\)) and floating point numbers that cover binary16, binary32, and binary64 (including non-finites) from [IEEE754]. With the tags defined with [RFC7049], the extended generic data model (Section 2.1 of [RFC8949]) adds unlimited-range integers (tag numbers 2 and 3, "bigint" in CDDL) as well as floating point values using the bases 2 (tag number 5, "bigfloat") and 10 (tag number 4, "decfrac").

This pre-defined number system has a number of limitations that are addressed in three of the tags discussed here:

* Tag number 30 allows the representation of rational numbers as a ratio of two integers: a numerator (usually written as the top part of a fraction), and a denominator (the bottom part), where both integers can be limited-range basic and unlimited-range integers. The mathematical value of a rational number is the numerator divided by the denominator. This tag can express all numbers that the extended generic data model of [RFC7049] can express, except for non-finites [IEEE754]; it also can express rational numbers that cannot be expressed with denominators that are a power of 2 or a power of 10.

For example, the rational number 1/3 is encoded:

```
d8 1e      ---- Tag 30
  82      ---- Array length 2
  01      ---- 1
  03      ---- 3
```

Many programming languages have built-in support for rational numbers or support for them is included in their standard libraries; tag number 30 is a way for these platforms to interchange these rational numbers in CBOR.

* Tag numbers 4 and 5 are limited in the range of the (base 10 or base 2) exponents by the limited-range integers in the basic generic data model. Tag numbers 264 and 265 are exactly equivalent to 4 and 5, respectively, but also allow unlimited-range integers as exponents. While applications for floating point numbers with exponents outside the CBOR basic integer range are limited, tags 264 and 265 allow unlimited roundtripping with other formats that allow very large or very small exponents, such as those JSON [RFC8259] can provide if the limitations of I-JSON [RFC7493] do not apply.
The tag numbers 268..270 extend these tags further by providing a way to express non-finites within a tag with this number. This does not increase the expressiveness of the data model (the non-finites can already be expressed using major type 7 floating point numbers), but does allow both finite and non-finite values to carry the same tag. In most applications, a choice that includes some of the three tags 30, 264, 265 for finite values and major type 7 floating point values for non-finites (as well as possibly other parts of the CBOR number system) will be the preferred solution.

This document suggests using the CDDL typenames defined in Figure 1 for the three most useful tag numbers in this section.

rational = #6.30([numerator: integer, denominator: integer .ne 0])
rationa_l_of<N,D> = #6.30([numerator: N, denominator: D])
; the value 1/3 can be notated as rational_of<1, 3>

extended_decfrac = #6.264([e10: integer, m: integer])
extended_bigfloat = #6.265([e2: integer, m: integer])

Figure 1: CDDL for extended arithmetic tags

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.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>byte string</td>
<td>uint8 Typed Array</td>
</tr>
<tr>
<td>65</td>
<td>byte string</td>
<td>uint16, big endian, Typed Array</td>
</tr>
<tr>
<td>66</td>
<td>byte string</td>
<td>uint32, big endian, Typed Array</td>
</tr>
<tr>
<td>67</td>
<td>byte string</td>
<td>uint64, big endian, Typed Array</td>
</tr>
<tr>
<td>68</td>
<td>byte string</td>
<td>uint8 Typed Array, clamped arithmetic</td>
</tr>
<tr>
<td>Tag</td>
<td>Content</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>byte string, uint16, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>byte string, uint32, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>byte string, uint64, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>byte string, sint8 Typed Array</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>byte string, sint16, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>byte string, sint32, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>byte string, sint64, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>byte string, (reserved)</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>byte string, sint16, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>byte string, sint32, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>byte string, sint64, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>byte string, IEEE 754 binary16, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>byte string, IEEE 754 binary32, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>byte string, IEEE 754 binary64, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>byte string, IEEE 754 binary128, big endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>byte string, IEEE 754 binary16, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>byte string, IEEE 754 binary32, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>byte string, IEEE 754 binary64, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>byte string, IEEE 754 binary128, little endian, Typed Array</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>array of two arrays*, Multi-dimensional Array, row-major order</td>
<td></td>
</tr>
<tr>
<td>1040</td>
<td>array of two arrays*, Multi-dimensional Array, column-major order</td>
<td></td>
</tr>
</tbody>
</table>
Table 6: Tag numbers defined for Arrays

7. Domain-Specific

(TO DO: Obtain permission to copy the definitions here; explain how tags 52 and 54 essentially obsolete 260/261.)
<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
<th>Reference</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>byte string</td>
<td>Binary UUID (Section 4.1.2 of [RFC4122])</td>
<td><a href="https://github.com/lucas-clemente/cbor-specs/blob/master/uuid.md">Reference</a></td>
<td>Lucas Clemente</td>
</tr>
<tr>
<td>38</td>
<td>array</td>
<td>Language-tagged string</td>
<td><a href="http://peteroupc.github.io/CBOR/langtags.html">Reference</a></td>
<td>Peter Occil</td>
</tr>
<tr>
<td>257</td>
<td>byte string</td>
<td>Binary MIME message</td>
<td><a href="http://peteroupc.github.io/CBOR/binarymime.html">Reference</a></td>
<td>Peter Occil</td>
</tr>
<tr>
<td>260</td>
<td>byte string</td>
<td>Network Address (IPv4 or IPv6 or MAC Address)</td>
<td><a href="http://www.employees.org/~ravir/cbor-network.txt">Reference</a></td>
<td>Ravi Raju</td>
</tr>
<tr>
<td>261</td>
<td>map</td>
<td>Network Address Prefix (IPv4 or IPv6 Address + Mask Length)</td>
<td><a href="https://github.com/toravir/CBOR-Tag-Specs/blob/master/networkPrefix.md">Reference</a></td>
<td>Ravi Raju</td>
</tr>
<tr>
<td>263</td>
<td>byte string</td>
<td>Hexadecimal string</td>
<td>[Reference](<a href="https://github.com/toravir/CBOR-Tag-Specs/blob/master(hexString.md)">https://github.com/toravir/CBOR-Tag-Specs/blob/master(hexString.md)</a></td>
<td>Ravi Raju</td>
</tr>
<tr>
<td>266</td>
<td>text string</td>
<td>Internationalized resource identifier (IRI)</td>
<td><a href="https://peteroupc.github.io/IRI.html">Reference</a></td>
<td>Peter Occil</td>
</tr>
<tr>
<td>267</td>
<td>text string</td>
<td>Internationalized resource identifier reference (IRI reference)</td>
<td><a href="https://peteroupc.github.io/IRI.html">Reference</a></td>
<td>Peter Occil</td>
</tr>
</tbody>
</table>

Table 7

7.1. Extended Time Formats

Additional tag definitions have been provided for date and time values.
Table 8: 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 2 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 2: 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 9.

(To Do: Obtain permission to copy the definitions here.)

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

Table 9: 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").

<table>
<thead>
<tr>
<th>Tag Number</th>
<th>Data Item</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>272</td>
<td>byte string</td>
<td>Non-UTF-8 CESU-8 string</td>
</tr>
<tr>
<td>273</td>
<td>byte string</td>
<td>Non-UTF-8 WTF-8 string</td>
</tr>
<tr>
<td>274</td>
<td>byte string</td>
<td>Non-UTF-8 MUTF-8 string</td>
</tr>
</tbody>
</table>

Table 10: Tag numbers for UTF-8 variants

9.  Application-specific

(To DO: Obtain permission to copy the definitions here.)
<table>
<thead>
<tr>
<th>Tag number</th>
<th>Tag content</th>
<th>Short Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>Lucas</td>
<td>Identifier</td>
<td>[<a href="https://github.com/lucas-clemente/cbor">https://github.com/lucas-clemente/cbor</a> specs/blob/master/id.md]</td>
</tr>
<tr>
<td>42</td>
<td>Volker</td>
<td>IPLD content</td>
<td>[<a href="https://github.com/ipld/cid-cbor/">https://github.com/ipld/cid-cbor/</a>]</td>
</tr>
<tr>
<td>103</td>
<td>Danilo</td>
<td>Geographic</td>
<td>[<a href="https://github.com/allthingstalk/cbor/">https://github.com/allthingstalk/cbor/</a> blob/master/ CBOR-Tag103-Geographic-Coordinates.md]</td>
</tr>
<tr>
<td>104</td>
<td>Multiple</td>
<td>Geographic Coordinate</td>
<td>[I-D.clarke-cbor-crs]</td>
</tr>
<tr>
<td>120</td>
<td>Danilo</td>
<td>Internet of Things Data Point</td>
<td>[<a href="https://github.com/allthingstalk/cbor/">https://github.com/allthingstalk/cbor/</a> nts.md CBOR-Tag120-Internet-of-Things-Data-Points.md]</td>
</tr>
<tr>
<td>258</td>
<td>Alfredo</td>
<td>Mathematical finite set</td>
<td>[<a href="https://github.com/input-output-hk/cbo">https://github.com/input-output-hk/cbo</a> r-sets- spec/blob/master/CBOR_SETS.md]</td>
</tr>
<tr>
<td>259</td>
<td>Shane</td>
<td>Map datatype with key-value operations (e.g. &quot;.get ()/.set()/.delete()&quot;)</td>
<td>[<a href="https://github.com/shanewholloway/js-c">https://github.com/shanewholloway/js-c</a> bor-explicit codec/blob/master/docs/ CBOR-259-spec-e maps.md]</td>
</tr>
</tbody>
</table>
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 "CBOR Tags" [IANA.cbor-tags], IANA has allocated the first to third tag in Table 12 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.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>65535</td>
<td>(none valid)</td>
<td>always invalid</td>
<td>draft-bormann-cbor-notable-tags, Section 10.1</td>
</tr>
<tr>
<td>4294967295</td>
<td>(none valid)</td>
<td>always invalid</td>
<td>draft-bormann-cbor-notable-tags, Section 10.1</td>
</tr>
<tr>
<td>18446744073709551615</td>
<td>(none valid)</td>
<td>always invalid</td>
<td>draft-bormann-cbor-notable-tags, Section 10.1</td>
</tr>
<tr>
<td>63</td>
<td>byte string</td>
<td>Encoded CBOR</td>
<td>draft-bormann-cbor-notable-tags, Section 2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sequence [RFC8742]</td>
<td></td>
</tr>
</tbody>
</table>

Table 12: 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
13.2. Informative References

[Cplusplus20]
International Organization for Standardization,
"Programming languages - C++", ISO/IEC ISO/IEC JTC1 SC22
WG21 N 4860, March 2020,

[I-D.clarke-cbor-crs]
Clarke, T. R., "Concise Binary Object Representation
(CBOR) Tag for Coordinate Reference System (CRS)
Specification", Work in Progress, Internet-Draft, draft-
clarke-cbor-crs-02, 17 March 2020,

[I-D.ietf-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-ietf-
cbor-time-tag-00, 19 May 2021,
<https://www.ietf.org/archive/id/draft-ietf-cbor-time-tag-
00.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,

[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-15, 1 February 2021,

[I-D.trammell-rains-protocol]
Trammell, B. and C. Fehlmann, "RAINS (Another Internet

Std 754-2019, DOI 10.1109/IEEESTD.2019.8766229,


Acknowledgements

(Many, TBD)

Contributors

Peter Occil
Email: poccil14 at gmail dot com

Peter Occil registered tags 30, 264, 265, 268-270 (Section 6.1), 38, 257, 266 and 267 (Section 7), and contributed much of the text about these tags in this document.

Jane Doe
To do

Further contributors will be listed here as text is added.

Please stay tuned.

Author’s Address

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

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

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.  Introduction  . . . . . . . . . . . . . . . . . . . . . . . .   3
  1.1.  Terminology . . . . . . . . . . . . . . . . . . . . . . .   3
2.  Objectives  . . . . . . . . . . . . . . . . . . . . . . . . .   3
3.  Time Format  . . . . . . . . . . . . . . . . . . . . . . . . .   4
  3.1.  Key 1 . . . . . . . . . . . . . . . . . . . . . . . . . .   5
  3.2.  Keys 4 and 5  . . . . . . . . . . . . . . . . . . . . . .   5
  3.3.  Keys -3, -6, -9, -12, -15, -18  . . . . . . . . . . . . .   5
  3.4.  Key -1: Time Scale  . . . . . . . . . . . . . . . . . . .   5
  3.5.  Clock Quality  . . . . . . . . . . . . . . . . . . . . . .   6
    3.5.1.  ClockClass (Key -2) . . . . . . . . . . . . . . . . .   6
    3.5.2.  ClockAccuracy (Key -4)  . . . . . . . . . . . . . . .   7
    3.5.3.  OffsetScaledLogVariance (Key -5)  . . . . . . . . . .   7
    3.5.4.  Uncertainty (Key -7)  . . . . . . . . . . . . . . . .   7
    3.5.5.  Guarantee (Key -8)  . . . . . . . . . . . . . . . . .   7
4.  Duration Format  . . . . . . . . . . . . . . . . . . . . . . .   8
5.  Period Format  . . . . . . . . . . . . . . . . . . . . . . . .   8
6.  CDDL typenames  . . . . . . . . . . . . . . . . . . . . . . . .  9
7.  IANA Considerations  . . . . . . . . . . . . . . . . . . . . .  9
8.  Security Considerations  . . . . . . . . . . . . . . . . . . .  9
9.  References  . . . . . . . . . . . . . . . . . . . . . . . . . . 10
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.

<table>
<thead>
<tr>
<th>Key</th>
<th>meaning</th>
<th>example usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>milliseconds</td>
<td>Java time</td>
</tr>
<tr>
<td>-6</td>
<td>microseconds</td>
<td>(old) UNIX time</td>
</tr>
<tr>
<td>-9</td>
<td>nanoseconds</td>
<td>(new) UNIX time</td>
</tr>
<tr>
<td>-12</td>
<td>picoseconds</td>
<td>Haskell time</td>
</tr>
<tr>
<td>-15</td>
<td>femtoseconds</td>
<td>(future)</td>
</tr>
<tr>
<td>-18</td>
<td>attoseconds</td>
<td>(future)</td>
</tr>
</tbody>
</table>

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.)

\[
\begin{align*}
t & = t - 2208988800 \\
\text{utc} & \quad \text{ntp} \\
\text{t} & = t + 315964819 \\
\text{tai} & \quad \text{gps}
\end{align*}
\]

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.

\[
\text{ClockQuality-group} = \{
\begin{align*}
? \text{ClockClass} & \Rightarrow \text{uint .size 1 ; PTP/RFC8575} \\
? \text{ClockAccuracy} & \Rightarrow \text{uint .size 1 ; PTP/RFC8575} \\
? \text{OffsetScaledLogVariance} & \Rightarrow \text{uint .size 2 ; PTP/RFC8575} \\
? \text{Uncertainty} & \Rightarrow \text{time/duration} \\
? \text{Guarantee} & \Rightarrow \text{time/duration}
\end{align*}
\}
\]

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.

\[
\text{enum} \quad \text{approx} \quad 48 + \left\lfloor \frac{\log_{10} \text{acc}}{2} \right\rfloor \approx 10
\]

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:

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

6. CDDL typtenames

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.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>map</td>
<td>[RFCthis] extended time</td>
</tr>
<tr>
<td>1002</td>
<td>map</td>
<td>[RFCthis] duration</td>
</tr>
<tr>
<td>1003</td>
<td>array</td>
<td>[RFCthis] period</td>
</tr>
</tbody>
</table>

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


9.2. Informative References


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
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.

Copyright Notice

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cat</td>
<td>String Concatenation</td>
</tr>
<tr>
<td>.plus</td>
<td>Numeric addition</td>
</tr>
<tr>
<td>.abnf</td>
<td>ABNF in CDDL (text strings)</td>
</tr>
<tr>
<td>.abnfb</td>
<td>ABNF in CDDL (byte strings)</td>
</tr>
<tr>
<td>.feature</td>
<td>Detecting feature use in extension points</td>
</tr>
</tbody>
</table>

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].
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]:

Bormann                  Expires 26 August 2021                 [Page 8]
### Table 2

<table>
<thead>
<tr>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.cat</td>
<td>[RFCthis]</td>
</tr>
<tr>
<td>.plus</td>
<td>[RFCthis]</td>
</tr>
<tr>
<td>.abnf</td>
<td>[RFCthis]</td>
</tr>
<tr>
<td>.abnfb</td>
<td>[RFCthis]</td>
</tr>
<tr>
<td>.feature</td>
<td>[RFCthis]</td>
</tr>
</tbody>
</table>

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,
8.2. Informative References


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
Additional Control Operators for CDDL

draft-ietf-cbor-cddl-control-07

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 were not yet ready at the time RFC 8610 was completed: .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 25 April 2022.

Copyright Notice

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

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

Code Components
1. Introduction

The Concise Data Definition Language (CDDL), standardized in [RFC8610], provides "control operators" as its main language extension point (Section 3.8 of [RFC8610]).

The present document defines a number of control operators that were not yet ready at the time RFC 8610 was completed:
### 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. "Tool" refers to tools along the lines of that described in Appendix F of [RFC8610]. Note also that the data model underlying CDDL provides for text strings as well as byte strings as two separate types, which are then collectively referred to as "strings".

The term ABNF in this specification stands for the combination of [RFC5234] and [RFC7405], i.e., the ABNF control operators defined by this document allow use of the case-sensitive extensions defined in [RFC7405].

### 2. Computed Literals

CDDL as defined in [RFC8610] does not have any mechanisms to compute literals. To cover a large part of the use cases, this specification adds three control operators: .plus for numeric addition, .cat for string concatenation, and .det for string concatenation with dedenting of both sides (target and controller).
For these operators, as with all control operators, targets and controllers are types. The resulting type is therefore formally a function of the elements of the cross-product of the two types. Not all tools may be able to work with non-unique targets or controllers.

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, i.e., rounding towards negative infinity).

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 CDDL group interval that gives a lower and an upper bound and optionally a tolerance. The parameter BASE allows the non-conflicting use of multiple of these interval groups in one map, by assigning different labels to the entries of the interval. rect combines two of these interval groups into a map, one group for the X dimension (using 0, 1, and 2 as labels) and one for Y dimension (using 3, 4, and 5 as labels).

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. 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 '
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 both arguments (target and controller) are independently _dedented_ before the concatenation takes place.

For the first rule in Figure 3, the result is equivalent to Figure 4.
For the purposes of this specification, we define dedenting as:

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

2. removing exactly that number of leading space characters from each line. For blank (blank 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.)

Occasionally, dedenting of only a single item is needed. This can be achieved by using this operator with an empty string, e.g., "" .det rhs or lhs .det "", which can in turn be combined with a .cat: in the construct lhs .cat ("" .det rhs), only rhs is dedented.

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, and/or by .det if there is indentation to be disposed of.

These points are combined into an example in Figure 5, which uses ABNF from [RFC3339] to specify one each of the CBOR tags defined in [RFC8943] and [RFC8949].
4. Features

Commonly, the kind of validation enabled by languages such as CDDL provides 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), as illustrated in Figure 6.

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

Figure 6: Providing explicit detail with .feature

Figure 7 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 7: Map extensibility with .feature

Figure 8 shows another example where .feature provides for type extensibility.

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

Figure 8: 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, as illustrated in Figure 9, using SenML [RFC8428] as the example data model used with both JSON and CBOR.

SenML-Record = {
  ; ...
  ? v => number
  ; ...
}

v = JC<v", 2

JC<J,C> = J .feature "json" / C .feature "cbor"

Figure 9: Describing variants with .feature

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 registry "CDDL Control Operators" of [IANA.cddl]:

Bormann                   Expires 25 April 2022                [Page 10]
Table 2: New control operators to be registered

6. Implementation Status

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

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 and 0.8.26 (double-handed .det).

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.

While both [RFC5234] and [RFC7405] state that security is truly believed to be irrelevant to the respective document, the use of formal description techniques cannot only simplify, but sometimes also complicate a specification. This can lead to security problems in implementations and in the specification itself. As with CDDL itself, ABNF should be judiciously applied, and overly complex (or "cute") constructions should be avoided.
8. References

8.1. Normative References


8.2. Informative References


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.

.det is an abbreviation for "dedenting cat", but Det is also the name of a German TV Cartoon character created in the 1960s.

Author’s Address

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

Phone: +49-421-218-63921
Email: cabo@tzi.org
On storing CBOR encoded items on stable storage
draft-ietf-cbor-file-magic-06

Abstract

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

Discussion Venues

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

Discussion of this document takes place on the CBOR Working Group mailing list (cbor@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/cbor/ (https://mailarchive.ietf.org/arch/browse/cbor/).

Source for this draft and an issue tracker can be found at https://github.com/cbor-wg/cbor-magic-number (https://github.com/cbor-wg/cbor-magic-number).

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 24 April 2022.
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 based upon the contents of the file.

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, starting in Mac OS, a resource fork was maintained that includes media type ("MIME type") information and therefore ideally never needs to know anything about the file.

But, many other systems do this by file extensions. Many common web servers derive the MIME-type information from 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. Additionally, both OpenOffice and MSOffice files are ZIP files of 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 possible method of enveloping data are presented: 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.ietf-cose-cbor-encoded-cert] would be an identified CBOR Protocol as well.

A major inspiration for this document is observing the mess in certain ASN.1 based systems where most files are PEM encoded, identified by the extension "pem", confusing public keys, private keys, certificate requests, and S/MIME content.
While these envelopes add information to how data conforming to CBOR Protocols are stored in files, there is no requirement that either type of envelope be transferred 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 are presently using a different (legacy) encoding scheme. The presence of the identifiable magic sequence signals that CBOR is being used as opposed to a legacy scheme.

1.1. Terminology

The term "diagnostic notation" refers to the human-readable notation for CBOR data items defined in Section 8 of [RFC8949] and Appendix G of [RFC8610].

The term CDDL (Concise Data Definition Language) refers to the language defined in [RFC8610].

1.2. Requirements for a Magic Number

A magic number is ideally a fingerprint that is unique to a CBOR protocol, present in the first few (small multiple of 4) 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 certain bytes, this can lead to ambiguities.

2. Protocol

There are two enveloping methods presented. Which one is to be used is up to the CBOR Protocol author to determine. Both use CBOR Tags in a way that results in a deterministic first 8 to 12 bytes.

2.1. The CBOR Protocol Specific Tag

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

The IANA policy for 4-byte CBOR Tags 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 needs to be allocated by the author of the CBOR Protocol. In order to be in the four-byte range, and so that there are no leading zeros, the value needs to be in the range 0x01000000 (decimal 16777216) to 0xFFFFFFFF (decimal 4294967295). It is further suggested to avoid values that have an embedded zero byte in the four bytes of their binary representation (e.g., 0x12003456).

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

For CBOR byte strings that happen to contain a representation that is described by a CoAP Content-Format Number (Section 12.3 of [RFC7252], Registry CoAP Content-Formats of [IANA.core-parameters]), a tag number has already been allocated in Section 5.2 (see Appendix A for details and examples).

2.2. Enveloping Method: CBOR Tag Wrapped

The CBOR Tag Wrapped method is appropriate for use with CBOR protocols that encode a single CBOR data item.

It starts with the Self-described CBOR tag, 55799, as described in Section 3.4.6 of [RFC8949].

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

This method 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.

Whether or not to remove the tags for specific further processing is a decision made by the CBOR Protocol specification.

2.2.1. Example

To construct an example without registering a new tag, we use the technique described in Appendix A to translate the Content-Format number registered for application/senml+cbor, the number 112, into the tag 1668546560+112 = 1668546672.

With this tag, the SenML-CBOR pack 

```
[{0: "current", 6: 3, 2: 1.5}]
```

would be enveloped as (in diagnostic notation):

```
55799(1668546672([{0: "current", 6: 3, 2: 1.5}])))
```

Or in hex:
In other words, the unique fingerprint for application/senml+cbor is composed of the 8 bytes d9d9f7da63740070 hex, after which the unadorned CBOR data (81... for the SenML data) is appended.

2.3. Enveloping Method: CBOR Tag Sequence

The CBOR Tag Sequence method is appropriate for use with CBOR Sequences as described in [RFC8742].

This method prepends a data item to the sequence to be tagged that consists of two tags nested around 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. As with the previous method, the use of a four-byte tag is encouraged that encodes without zero bytes.

3. The encoded three byte CBOR byte string containing 0x42_4F_52.

The first part identifies the file as being a CBOR Sequence, and does so with all the desirable properties explained in Section 3.4.6 of [RFC8949]. Specifically, it does not appear 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 represented as a constant byte sequence 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three byte sequence 0x42_4f_52 ('BOR' in diagnostic notation). This is the data item that is being 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.

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

3. Advice to Protocol Developers

This document introduces a choice between wrapping a single CBOR data item into a (pair of) identifying CBOR tags, or prepending an identifying encoded CBOR data item (which in turn contains a pair of identifying CBOR tags) to a CBOR Sequence (which might be single data item).

Which should a protocol designer use?

In this discussion, one assumes that there is an object stored in a file, perhaps specified by a system operator in a configuration file.

For example: a private key used in COSE operations, a public key/certificate in C509 or CBOR format, a recorded sensor reading stored for later transmission, or a COVID vaccination certificate that needs to be displayed in QR code form.

Both the CBOR Tag Sequence and the wrapped tag can be trivially removed by an application before sending the CBOR content out on the wire.

The CBOR Tag Sequence is a little bit easier to remove as in most cases, CBOR parsers will return it as a unit, and then return the actual CBOR item, which could be anything at all, and could include CBOR tags that _do_ need to be sent on wire.

On the other hand, having the CBOR Tag Sequence in the file requires that all programs that expect to examine that file are able to skip what appears to be a CBOR item with two tags nested around a three-byte byte string. Programs which might not expect the CBOR Tag Sequence, but which would operate without a problem would include any program that expects to process CBOR Sequences from the file.

As an example of where there was a problem with previous security systems, "PEM" format certificate files grew to be able to contain multiple certificates by simple concatenation. The PKCS1 format could also contain a private key object followed by a one or more
certificate objects: but only when in PEM format. Annoyingly, when in binary DER format (which like CBOR is self-delimiting), concatenation of certificates was not compatible with most programs as they did not expect to read more than one item in the file.

The use of CBOR Tag Wrapped format is easier to retrofit to an existing format with existing and unchangeable on-disk format for a single CBOR data item. This new sequence of tags is expected to be trivially ignored by many existing programs when reading CBOR from disk, even if the program only supports decoding a single data item (and not a CBOR sequence). But, a naive program might also then transmit the additional tags across the network. Removing the CBOR Tag Wrapped format requires knowledge of the two tags involved. Other tags present might be needed.

For a representation matching a specific media-type that is carried in a CBOR byte string, the byte string head will already have to be removed for use as such a representation, so it should be easy to remove the enclosing tag heads as well. This is of particular interest with the pre-defined tags provided by Appendix A for media-types with CoAP Content-Format numbers.

Here are some considerations in the form of survey questions:

3.1. Is the on-wire format new?

If the on-wire format is new, then it could be specified with the CBOR Tag Wrapped format if the extra eight bytes are not a problem. The disk format is then identical to the on-wire format.

If the eight bytes are a problem on the wire (and they often are if CBOR is being considered), then the CBOR Tag Sequence format should be adopted for on-disk storage.

3.2. Can many items be trivially concatenated?

If the programs that read the contents of the file already expect to process all of the CBOR data items in the file (not just the first), then the CBOR Tag Sequence format may be easily retrofitted.

The program involved may throw errors or warnings on the CBOR Tag Sequence if they have not yet been updated, but this may not be a problem. If it is, then consideration should be given to CBOR Tag Wrapped.

If only one item is ever expected in the file, the use of CBOR Tag Sequence may present an implementation hurdle to programs that previously just read a single data item and used it.
3.3. Are there tags at the start?

If the Protocol expects to use other tags values at the top-level, then it may be easier to explain if the CBOR Tag Sequence format is used.

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 exfiltrate or corrupt them.

5. IANA Considerations

Section 5.1 documents the allocation that was done for a CBOR tag to be used in a CBOR sequence to identify the sequence (an example for using this tag is found in Appendix B). Section 5.2 allocates a CBOR tag for each actual or potential CoAP Content-Format number (examples are in Appendix A).

5.1. CBOR Sequence Tag

IANA has allocated tag 55800 as the tag for the CBOR Tag Sequence Enveloping Method.

This tag is from the First Come/First Served area.

The value has been picked to have properties similar to the 55799 tag (Section 3.4.6 of [RFC8949]).

The hexadecimal representation of the encoded tag head is: 0xd9_d9_f8.

This is not valid UTF-8: the first 0xd9 introduces a three-byte sequence in 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 Section 3.4.6 of [RFC8949] has already determined that no valid CBOR encodings result in valid UTF-16.

Data Item:
  tagged byte string

Semantics:
  indicates that the file contains CBOR Sequences

5.2. CBOR Tags for CoAP Content-Format Numbers

IANA is requested to allocate the tag numbers 1668546560 (0x63740000) to 1668612095 (0x6374FFFF) as follows:

Data Item:
  byte string

Semantics:
  for each tag number NNNNNNNN, the representation of content-format (RFC7252) NNNNNNNN-1668546560

Reference:
  RFCthis

6. References

6.1. Normative References


6.2. Informative References

[I-D.ietf-cose-cbor-encoded-cert]
Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and
M. Furuhed, "CBOR Encoded X.509 Certificates (C509
Certificates)", Work in Progress, Internet-Draft, draft-
ietf-cose-cbor-encoded-cert-02, 12 July 2021,
<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-
encoded-cert-02.txt>.

[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-10, 7 June
2021, <https://www.ietf.org/archive/id/draft-ietf-rats-
eat-10.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-19, 20
October 2021, <https://www.ietf.org/archive/id/draft-ietf-
sacm-coswid-19.txt>.

[IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE)

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO
10646", RFC 2781, DOI 10.17487/RFC2781, February 2000,

Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,

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

[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,
Appendix A. CBOR Tags for CoAP Content Formats

Often, there is a need to identify a media type (or content type, i.e., media type optionally used with parameters) that describes a byte string in a CBOR data item.

Section 5.10.3 of [RFC7252] defines the concept of a Content-Format, which is a short 16-bit unsigned integer that identifies a specific content type (media type plus optionally parameters), optionally together with a content encoding.

This specification allocates CBOR tag numbers 1668546560 (0x63740000) to 1668612095 (0x6374FFFF) for the tagging of representations of specific content formats. The tag content tagged with tag number NNNNNNNNNN (in above range) is a byte string that is to be interpreted as a representation of the content format NNNNNNNNNN-1668546560.

A.1. Content-Format Tag Examples

Registry Content-Formats of [IANA.core-parameters] defines content formats that can be used as examples:

* Content-Format 432 stands for media type application/td+json (no parameters). The corresponding tag number is 1668546992 (i.e., 1668546560+432).

So the following CDDL snippet can be used to identify application/td+json representations:

\[\text{td-json} = \#6.1668546992(\text{bstr})\]

Note that a byte string is used as the type of the tag content, because a media type representation in general can be any byte string.

* Content-Format 11050 stands for media type application/json in deflate encoding.

The corresponding tag number is 1668557610 (i.e., 1668546560+11050).

So the following CDDL snippet can be used to identify application/json representations compressed in deflate encoding:

\[\text{json-deflate} = \#6.1668557610(\text{bstr})\]
The byte string is appropriate here as the type for the tag content, because the compressed form is an instance of a general byte string.

Appendix B. 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 a CBOR TAG Sequence.

In diagnostic notation:

55800(1330664270(h’424F52’))

Or in hex:

D9 D9F8 # tag(55800)
DA 4F50534E # tag(1330664270)
   43 # bytes(3)
   424F52 # "BOR"

Appendix C. Changelog
Acknowledgements

The CBOR WG brainstormed this protocol on January 20, 2021 via a number of productive email exchanges on the mailing list.

Contributors

Josef ’Jeff’ Sipek
Email: jeffpc@josefsipek.net

Authors’ Addresses

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca

Carsten Bormann
Universität Bremen Tzi
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org
CBOR tags for IPv4 and IPv6 addresses and prefixes
draft-ietf-cbor-network-addresses-13

Abstract

This specification defines two CBOR Tags for use with IPv6 and IPv4 addresses and prefixes.

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

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 25 April 2022.

Copyright Notice

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

[RFC8949] defines a number of CBOR Tags for common items. Tags 260 and 261 were later defined in drafts listed with IANA [IANA.cbor-tags]. These tags were intended to cover addresses (260) and prefixes (261). Tag 260 distinguishes between IPv6, IPv4, and MAC [RFC7042] addresses only through the length of the byte string, making it impossible, for example, to drop trailing zeros in the encoding of IP addresses. Tag 261 was not documented well enough for use.
This specification defines tags 54 and 52 achieving an explicit indication of IPv6 or IPv4 by the tag number. These new tags are intended to be used in preference to tags 260 and 261. They provide formats for IPv6 and IPv4 addresses, prefixes, and addresses with prefixes, achieving an explicit indication of IPv6 or IPv4. The prefix format omits trailing zeroes in the address part. (Due to the complexity of testing, the value of omitting trailing zeros for the pure address format was considered non-essential and support for that is not provided in this specification.) This specification does not deal with MAC addresses (Section 2 of [RFC7042]) such as they are used for Ethernet.

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

3.1. Three Forms

3.1.1. Addresses

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

This form is called the Address Format.

3.1.2. Prefixes

When applied to an array that starts with an unsigned integer, they represent a CIDR-style prefix of that length.

When the Address Format (i.e., 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.

This form is called the Prefix Format.
3.1.3. Interface Definition

When applied to an array that starts with a byte string, which stands for an IP address, followed by an unsigned integer giving the bit length of a prefix built out of the first length bits of the address, they represent information that is commonly used to specify both the network prefix and the IP address of an interface.

The length of the byte string is always 16 bytes (for IPv6) and 4 bytes (for IPv4).

This form is called the Interface Format.

Interface Format definitions support an optional third element to the array, which is to be used as the IPv6 Link-Local zone identifier from Section 6 of [RFC4007]; for symmetry this is also provided for IPv4 as in [RFC4001] and [RFC6991]. The zone identifier may be an integer, in which case it is to be interpreted as the interface index. It may be a text string, in which case it is to be interpreted as an interface name.

As explained in [RFC4007] the zone identifiers are strictly local to the node. They are useful for communications within a node about connected addresses (for instance, where a link-local peer is discovered by one daemon, and another daemon needs to be informed). They may also have utility in some management protocols.

In the cases where the Interface Format is being used to represent only an address with a zone identifier, and no interface prefix information, then the prefix length may be replaced with the CBOR "null" (0xF6).

3.2. IPv6

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

An IPv6 address is to be encoded as a sixteen-byte byte string (Section 3.1 of [RFC8949], major type 2), enclosed in Tag number 54.

For example:

54(h'20010db81234deedbeefcafefacefeed')

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. See Section 4 for the detailed construction of the second element.
For example:

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

An IPv6 address combined with a prefix length, such as being used for configuring an interface, is to be encoded as a two element array, with the (full-length) IPv6 address first and the length of the associated network the prefix next; a third element can be added for the zone identifier.

For example:

54([h'20010db81234deedbeefcafefacefeed', 56])

The address-with-prefix form can be reliably distinguished from the prefix form only in the sequence of the array elements.

Some example of a link-local IPv6 address with a 64-bit prefix:

54([h'fe80000000000020202fffffffe030303', 64, 'eth0'])

with a numeric zone identifier:

54([h'fe80000000000020202fffffffe030303', 64, 42])

An IPv6 link-local address without a prefix length:

54([h'fe80000000000020202fffffffe030303', null, 42])

Zone identifiers may be used with any kind of IP address, not just Link-Local addresses. In particular, they are valid for multicast addresses, and there may still be some significance for Globally Unique Addresses (GUA).

3.3. IPv4

IANA has allocated tag 52 for IPv4 uses. (This is the ASCII code for ‘4’.)

An IPv4 address is to be encoded as a four-byte byte string (Section 3.1 of [RFC8949], major type 2), enclosed in Tag number 52.

For example:

52(h’c0000201’)

Richardson & Bormann Expires 25 April 2022 [Page 5]
An IPv4 prefix, such as 192.0.2.0/24 is to be encoded as a two element array, with the length of the prefix first. See Section 4 for the detailed construction of the second element.

For example:

52([24, h’c00002’])

An IPv4 address combined with a prefix length, such as being used for configuring an interface, is to be encoded as a two element array, with the (full-length) IPv4 address first and the length of the associated network the prefix next; a third element can be added for the zone identifier.

For example, 192.0.2.1/24 is to be encoded as a two element array, with the length of the prefix (implied 192.0.2.0/24) last.

52([h’c0000201’, 24])

The address-with-prefix form can be reliably distinguished from the prefix form only in the sequence of the array elements.

4. Tag validity

This section discusses when a tag 54 or tag 52 is valid (Section 5.3.2 of [RFC8949]). As with all CBOR tags, validity checking can be handled in a generic CBOR library or in the application. A generic CBOR library needs to document whether and how it handles validity checking.

The rule ip-address-or-prefix in Figure 1 shows how to check the overall structure of these tags and their content, the ranges of integer values, and the lengths of byte strings. An instance of tag 52 or 54 is valid if it matches that rule and, for ipv6-prefix and ipv4-prefix, the considerations of Sections 4.2 and 4.3.

4.1. Deterministic Encoding

The tag validity rules, combined with the rules in Section 4.2.1 of [RFC8949], lead to deterministic encoding for tags 54 and 52 and require no further Additional Deterministic Encoding Considerations as per Section 4.2.2 of [RFC8949].

4.2. Encoder Considerations for Prefixes

For the byte strings used as the second element in the array representing a prefix:
(1) An encoder MUST set any unused bytes, and any unused bits in the final byte, if any, to zero. Unused bytes/bits are bytes/bits that are not covered by the prefix length given. So for example, 2001:db8:1230::/44 MUST be encoded as:

54([44, h'20010db81230'])

even though variations like:

54([44, h'20010db81233'])
54([44, h'20010db8123f'])
54([44, h'20010db8123012'])

start with the same 44 bits, but are not valid.

(Analogous examples can be constructed for IPv4 prefixes.)

(2) An encoder MUST then omit any right-aligned (trailing) sequence of bytes that are all zero.

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

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

4.3. Decoder Considerations for Prefixes

A decoder MUST check that all unused bits encoded in the byte string ipv6-prefix-bytes/ipv4-prefix-bytes, i.e., the bits to the right of the prefix length, are zero.

A decoder MUST also check that the byte string does not end in a zero byte.

Since encoders are required to remove zero-valued trailing bytes, 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 an example, ::/128 is encoded as

54([128, h''])

4.3.1. Example implementation

A recommendation for prefix decoder implementations is to first create an array of 16 (or 4) zero bytes.
Then taking whichever is smaller between (a) the length of the included byte-string, and (b) the number of bytes covered by the prefix-length rounded up to the next multiple of 8: fail if that number is greater than 16 (or 4), and then copy that many bytes from the byte-string into the byte array.

Finally, looking at the number of unused bits in the last byte (if any) of the range covered by the prefix length, check that any unused bits in the byte string are zero:

```
unused_bits = (8 - (prefix_length_in_bits % 8)) % 8;
if (length_in_bytes > 0 &&
    (address_bytes[length_in_bytes - 1] & ~(0xFF << unused_bits))
        != 0)
    fail();
```

5. CDDL

For use with CDDL [RFC8610], the typenames defined in Figure 1 are recommended:
6. Security Considerations

This document provides an CBOR encoding for IPv4 and IPv6 address information. Any applications using these encodings will need to consider the security implications of these data in their specific context. For example, 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.

Applications need to check the validity (Section 4) of a tag before acting on any of its contents. If the validity checking is not done in the generic CBOR decoder, it needs to be done in the application; in any case it needs to be done before the tag is transformed into a platform-specific representation that could conceal validity errors.
The right-hand bits of the prefix, after the prefix-length, are set to zero by this protocol. (Otherwise, 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 is detected by tag validity checking, and can also be detected by examination of the raw protocol bytes.)

7. IANA Considerations

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

7.1. Tag 54 - IPv6

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

7.2. Tag 52 - IPv4

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

7.3. Tags 260 and 261

IANA is requested to add the note "DEPRECATED in favor of 52 and 54 for IP addresses" to registrations 260 and 261

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Changelog

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

* 03
* 02
* 01 added security considerations about covert channel

Acknowledgements

Roman Danyliw, Donald Eastlake, Ben Kaduk, Barry Leiba, and Éric Vyncke reviewed the document and provided suggested text. Jürgen Schönwälder helped finding the history of IPv4 zone identifiers.

Authors' Addresses

Richardson & Bormann Expires 25 April 2022 [Page 11]
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


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 13 August 2021.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 2
   1.1. Terminology ............................................ 3
2. Packed CBOR ................................................ 4
   2.1. Packing Tables ......................................... 4
   2.2. Referencing Shared Items ............................... 4
   2.3. Referencing Affix Items ............................... 5
   2.4. Discussion ............................................. 7
3. Table Setup ................................................. 8
   3.1. Basic Packed CBOR ....................................... 8
4. IANA Considerations ......................................... 9
5. Security Considerations ...................................... 10
6. References ................................................. 10
   6.1. Normative References .................................... 10
   6.2. Informative References .................................. 11
Appendix A. Examples ........................................... 12
Acknowledgements ............................................. 16
Author’s Address ............................................. 16

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.
### 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.
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.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>integer, text string, byte string, array, map</td>
<td>Packed CBOR: shared/prefix</td>
<td>draft-ietf-cbor-packed</td>
</tr>
<tr>
<td>225-255</td>
<td>text string, byte string, array, map</td>
<td>Packed CBOR: prefix</td>
<td>draft-ietf-cbor-packed</td>
</tr>
<tr>
<td>28704-32767</td>
<td>text string, byte string, array, map</td>
<td>Packed CBOR: prefix</td>
<td>draft-ietf-cbor-packed</td>
</tr>
<tr>
<td>1879052288-2147483647</td>
<td>text string, byte string, array, map</td>
<td>Packed CBOR: prefix</td>
<td>draft-ietf-cbor-packed</td>
</tr>
<tr>
<td>216-223</td>
<td>text</td>
<td>Packed</td>
<td>draft-ietf-cbor-packed</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Packed CBOR: shared</td>
<td>draft-ietf-cbor-packed</td>
</tr>
</tbody>
</table>

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, "Concise Binary Object Representation (CBOR) Simple Values",

IANA, "Concise Binary Object Representation (CBOR) Tags",

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174,

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,

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949,
DOI 10.17487/RFC8949, December 2020,

6.2. Informative References


Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322,
DOI 10.17487/RFC7322, September 2014,

Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020,
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.

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

```json
{
    "name": "MyLED",
    "interactions": [
        {
            "links": [
                {
                    "mediaType": "application/json"
                }
            ],
            "outputData": {
                "valueType": {
                    "type": "number"
                },
                "name": "rgbValueRed",
                "writable": true,
                "@type": [
                    "Property"
                ]
            }
        },
        {
            "links": [
                {
                    "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": [
  {
    "mediaType": "application/json"
  }
],
"outputData": {
  "valueType": {
    "type": "number"
  }
},
"name": "rgbValueWhite",
"writable": true,
"@type": [
  "Property"
]
Figure 4: Example original CBOR data item
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.

Acknowledgements

Carsten Bormann
Universität Bremen TZI
Postfach 330440

Bormann  Expires 13 August 2021
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) can work well for CBOR encoded data items, 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


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 14 February 2022.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 2
   1.1. Terminology ............................................. 3
2. Packed CBOR ................................................... 4
   2.1. Packing Tables .......................................... 4
   2.2. Referencing Shared Items ............................... 4
   2.3. Referencing Affix Items ................................ 5
   2.4. Discussion ............................................. 7
3. Table Setup .................................................... 8
   3.1. Basic Packed CBOR ....................................... 9
4. IANA Considerations ........................................... 10
5. Security Considerations ...................................... 11
6. References .................................................... 11
   6.1. Normative References ................................... 11
   6.2. Informative References ................................ 12
Appendix A. Examples ............................................. 13
Acknowledgements ................................................ 17
Author’s Address ................................................. 18

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.

Packed CBOR is defined in two parts: Referencing packing tables (Section 2) and setting up packing tables (Section 3).

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, and, in the HTML and PDF versions, subtraction and negation are rendered as a hyphen ("--", as are various dashes).

2. Packed CBOR

This section describes the packing tables, their structure, and how they are referenced.

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. Each of the tables is indexed by an unsigned integer (starting from 0), which may be computed from information in tags and their content as well as from CBOR simple values.

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.
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.
Table 2: Referencing Prefix Values

<table>
<thead>
<tr>
<th>prefix reference</th>
<th>table index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag 6(suffix)</td>
<td>0</td>
</tr>
<tr>
<td>Tag 225-255(suffix)</td>
<td>1-31</td>
</tr>
<tr>
<td>Tag 28704-32767(suffix)</td>
<td>32-4095</td>
</tr>
<tr>
<td>Tag 1879052288-2147483647(suffix)</td>
<td>4096-268435455</td>
</tr>
</tbody>
</table>

Table 3: Referencing Suffix Values

<table>
<thead>
<tr>
<th>suffix reference</th>
<th>table index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag 216-223(prefix)</td>
<td>0-7</td>
</tr>
<tr>
<td>Tag 27647-28671(prefix)</td>
<td>8-1023</td>
</tr>
<tr>
<td>Tag 1811940352-1879048191(prefix)</td>
<td>1024-67108863</td>
</tr>
</tbody>
</table>

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 is not 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.

Different strategies for decoding/consuming Packed CBOR are available. For example:
* the decoder can decode and unpack the packed item, presenting an unpacked data item to the application. In this case, the onus of dealing with loops is on the decoder. (This strategy generally has the highest memory consumption, but also the simplest interface to the application.) Besides avoiding getting stuck in a reference loop, the decoder will need to control its resource allocation, as data items can "blow up" during unpacking.

* the decoder can be oblivious of Packed CBOR. In this case, the onus of dealing with loops is on the application, as is the entire onus of dealing with Packed CBOR.

* hybrid models are possible, for instance: The decoder builds a data item tree directly from the Packed CBOR as if it were oblivious, but also provides accessors that hide (resolve) the packing. In this specific case, the onus of dealing with loops is on the accessors.

In general, loop detection can be handled in a similar way in which loops of symbolic links are handled in a file system: A system wide limit (often 31 or 40 indirections for symbolic links) is applied to any reference chase.

ISSUE: The present specification does nothing to help with the packing of 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. Note that, without this information, a data item that uses such a static dictionary can be decoded at the CBOR level, but not fully unpacked. The table setup mechanisms provided by this document are defined in such a way that an unpacker can at least recognize if this is the case.
* 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.

For table setup, the present specification only defines a single tag,
which operates by 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 desirable
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-item], [*prefix-item],
                           [*suffix-item], rump])
rump = any
prefix-item = any
suffix-item = any
shared-item = any

Figure 1: Packed CBOR in CDDL

(This assumes the allocation of tag number 51 for this tag.)

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.
Packed item references in the newly constructed (low-numbered) parts of the table need to be interpreted in the number space of that table (which includes the, now higher-numbered inherited parts), while references in any existing, inherited (higher-numbered) part continue to use the (more limited) number space of the inherited table.

4. IANA Considerations

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

<table>
<thead>
<tr>
<th>Tag (for shared); text string, byte string, array, map (for prefix)</th>
<th>Data Item</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 integer</td>
<td>Packed CBOR: shared/prefix</td>
<td>draft-ietf-cbor-packed</td>
<td></td>
</tr>
<tr>
<td>225-255 text string, byte string, array, map</td>
<td>Packed CBOR: prefix</td>
<td>draft-ietf-cbor-packed</td>
<td></td>
</tr>
<tr>
<td>28704-32767 text string, byte string, array, map</td>
<td>Packed CBOR: prefix</td>
<td>draft-ietf-cbor-packed</td>
<td></td>
</tr>
<tr>
<td>1879052288-2147483647 text string, byte string, array, map</td>
<td>Packed CBOR: prefix</td>
<td>draft-ietf-cbor-packed</td>
<td></td>
</tr>
<tr>
<td>216-223 text</td>
<td>Packed</td>
<td>draft-ietf-cbor-packed</td>
<td></td>
</tr>
</tbody>
</table>
Table 4: Values for Tag Numbers

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-15</td>
<td>Packed CBOR: shared</td>
<td>draft-ietf-cbor-packed</td>
</tr>
</tbody>
</table>

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, "Concise Binary Object Representation (CBOR) Simple Values",

IANA, "Concise Binary Object Representation (CBOR) Tags",

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174,
DOI 10.17487/RFC8174, May 2017,

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,

Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949,
DOI 10.17487/RFC8949, December 2020,

6.2. Informative References


Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049,

Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322,
DOI 10.17487/RFC7322, September 2014,

Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020,
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.

```json
{  
  "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
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.

```json
{
  "name": "MyLED",
  "interactions": [
    {
      "links": [
        {
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueRed",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueRed",
      "writable": true,
      "@type": [
        "Property"
      ]
    }
  ]
}
```
"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": [ {
  "mediaType": "application/json"
} ],
"outputData": {
  "valueType": {
    "type": "number"
  },
  "name": "rgbValueWhite",
  "writable": true,
  "@type": [ "Property" ]
},
"links": [ {
  "href": "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueYellow",
  "mediaType": "application/json"
} ],
"outputData": {
  "valueType": {
    "type": "number"
  },
  "name": "rgbValueYellow",
  "writable": true,
  "@type": [ "Property" ]
},
"links": [ {
  "mediaType": "application/json"
} ],
"outputData": {
  "valueType": {
    "type": "number"
  },
  "name": "rgbValueRed",
  "writable": true,
  "@type": [ "Property" ]
},
"links": [ {
  "mediaType": "application/json"
} ],
"outputData": {
  "valueType": {
    "type": "number"
  },
  "name": "rgbValuePink",
  "writable": true,
  "@type": [ "Property" ]
},
"links": [ {
  "href": "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueOrange",
  "mediaType": "application/json"
} ],
"outputData": {
  "valueType": {
    "type": "number"
  },
  "name": "rgbValueOrange",
  "writable": true,
  "@type": [ "Property" ]
},
"links": [ {
  "mediaType": "application/json"
} ],
"outputData": {
  "valueType": {
    "type": "number"
  },
  "name": "rgbValueGray",
  "writable": true,
  "@type": [ "Property" ]
}
"writable": true,
"@type": [
  "Property"
]
},
{ "links": [
  {
    "href":
      "mediaType": "application/json"
  }
],
  "outputData": {
    "valueType": {
      "type": "boolean"
    }
  },
  "name": "ledOnOff",
  "writable": true,
  "@type": [
    "Property"
  ]
},
{ "links": [
  {
    "href":
      "http://192.168.1.103:8445/wot/thing/MyLED/colorTemperatureChanged",
      "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": [
   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")}),
   simple(0): "rgbValueWhite")},
   simple(0): 227("ledOnOff")},
229({simple(2): [{simple(3): 226("colorTemperatureChanged"),
   simple(4): simple(5)}],
   simple(0): 227("colorTemperatureChanged")},
   "@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
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

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

\[
X \times 40 + Y
\]
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

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

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 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:
### 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:

```json
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
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.
Table 2: Values for New Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD111</td>
<td>byte string or array or map</td>
<td>object identifier (BER encoding)</td>
</tr>
<tr>
<td>TBD110</td>
<td>byte string or array or map</td>
<td>relative object identifier (BER encoding); SDNV [RFC6256] sequence</td>
</tr>
<tr>
<td>TBD112</td>
<td>byte string or array or map</td>
<td>object identifier (BER encoding), relative to 1.3.6.1.4.1</td>
</tr>
</tbody>
</table>

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.

Table 3: New CDDL Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.sdnv</td>
<td>[this document, Section 5]</td>
</tr>
<tr>
<td>.sdnvseq</td>
<td>[this document, Section 5]</td>
</tr>
<tr>
<td>.oid</td>
<td>[this document, Section 5]</td>
</tr>
</tbody>
</table>

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


9.2. Informative References


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
Concise Binary Object Representation (CBOR) Tags for Object Identifiers
draft-ietf-cbor-tags-oid-08

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 22 November 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the
document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
  1.1. Terminology ............................................. 3
2. Object Identifiers ........................................... 4
   2.1. Requirements on the byte string being tagged ............ 5
   2.2. Preferred Serialization Considerations ................. 6
   2.3. Discussion ............................................ 6
3. Basic Examples ................................................ 7
   3.1. Encoding of the SHA-256 OID ............................. 7
   3.2. Encoding of a MIB Relative OID .......................... 7
4. Tag Factoring with Arrays and Maps .......................... 8
   4.1. Preferred Serialization Considerations ................. 8
   4.2. Tag Factoring Example: X.500 Distinguished Name .......... 9
5. CDDL Control Operators ....................................... 10
6. CDDL typenames ............................................... 11
7. IANA Considerations .......................................... 11
   7.1. CBOR Tags .............................................. 11
   7.2. CDDL Control Operators ................................ 12
8. Security Considerations ...................................... 12
9. References ..................................................... 13
   9.1. Normative References ................................... 13
   9.2. Informative References ................................ 14
Appendix A. Change Log ........................................... 15
   A.1. Changes from -06 to -07 ................................ 15
   A.2. Changes from -05 to -06 ................................ 15
   A.3. Changes from -04 to -05 ................................ 15
   A.4. Changes from -03 to -04 ................................ 15
   A.5. Changes from -02 to -03 ................................ 15
   A.6. Changes from -01 to -02 ................................ 15
   A.7. Changes from -00 to -01 ................................ 15
   A.8. Changes from -07 (bormann) to -00 (ietf) ............... 16
   A.9. Changes from -06 to -07 ................................ 16
   A.10. Changes from -05 to -06 ................................ 16
   A.11. Changes from -04 to -05 ................................ 16
   A.12. Changes from -03 to -04 ................................ 16
   A.13. Changes from -02 to -03 ................................ 16
Acknowledgments .................................................. 17
Contributors ....................................................... 17
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, and a third one to enable a common optimization. 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 verb "to tag (something)" is used to express the construction of a CBOR tag with the object (something) as the tag content and a tag number indicated elsewhere in the sentence (for instance in a "with" clause, or by the shorthand "an NNN tag" for "a tag with tag number NNN"). The term "SDNV" (Self-Delimiting Numeric Value) is used as defined in [RFC6256], with the additional restriction detailed in Section 2.1 (no leading zeros).
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:

\[ X \times 40 + Y \]

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, and it is very common for companies to use self-assigned numbers under the arc "1.3.6.1.4.1" (IANA Private Enterprise Number OID, [IANA.enterprise-numbers]) that adds 5 fixed bytes to an encoded OID value, this specification defines three tags, collectively called the "OID tags" here:
Tag number TBD111: used to tag a byte string as the [X.690] encoding of an absolute object identifier (simply "object identifier" or "OID").

Tag number TBD110: used to tag 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 (or a more application-specific tag can be created for such an application).

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 with 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
regular expression is invalid.

2.2. Preferred Serialization Considerations

For an absolute OID with a prefix of "1.3.6.1.4.1", representations
with both the TBD111 and TBD112 tags are applicable, where the
representation with TBD112 will be five bytes shorter (by leaving out
the prefix h'2b06010401' from the enclosed byte string). This
specification makes that shorter representation the preferred
serialization (see Sections 3.4 and 4.1 of [RFC8949]). Note that
this also implies that the Core Deterministic Encoding Requirements
(Section 4.2.1 of [RFC8949]) require the use of TBD112 tags instead
of TBD111 wherever that is possible.

2.3. 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 # 0b10_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
This relative OID saves seven bytes compared to the full OID encoding.

4. Tag Factoring with Arrays and Maps

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

When the tag content of an OID tag is an array, this 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 the tag content of a TBD111 tag is an array, 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 the tag content of an OID tag is a map, this means that a tag with the same tag number 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 byte strings. All such byte strings are then considered OIDs.

4.1. Preferred Serialization Considerations

Where tag factoring with tag number TBD111 is used, some OIDs enclosed in the tag may be encoded in a shorter way by using tag number TBD112 instead of encoding an unadorned byte string. This remains the preferred serialization (see also Section 2.2). However, this specification does not make the presence or absence of tag factoring a preferred serialization; application protocols can define where tag factoring is to be used or not (and will need to do so if they have deterministic encoding requirements).
4.2. Tag Factoring Example: X.500 Distinguished Name

Consider the X.500 distinguished name:

<table>
<thead>
<tr>
<th>Attribute Types</th>
<th>Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>c (2.5.4.6)</td>
<td>US</td>
</tr>
<tr>
<td>l (2.5.4.7)</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>s (2.5.4.8)</td>
<td>CA</td>
</tr>
<tr>
<td>postalCode (2.5.4.17)</td>
<td>90013</td>
</tr>
<tr>
<td>street (2.5.4.9)</td>
<td>532 S Olive St</td>
</tr>
<tr>
<td>businessCategory (2.5.4.15)</td>
<td>Public Park</td>
</tr>
<tr>
<td>buildingName</td>
<td>Pershing Square</td>
</tr>
<tr>
<td>(0.9.2342.19200300.100.1.48)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Example X.500 Distinguished Name

Table 1 has four "relative distinguished names" (RDNs). The country (first) and street (third) 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
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

Concise Data Definition Language (CDDL [RFC8610]) 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.

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

Figure 7: Using .sdnvseq
```

```plaintext
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, the typenames defined in Figure 9 are recommended:

```plaintext
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 in the 1+1 byte space (24..255) of the CBOR tags registry [IANA.cbor-tags] the CBOR tag numbers in Table 2, with the present document as the specification reference.
### Table 2: New Tag Numbers

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD111</td>
<td>byte string or array or map</td>
<td>object identifier (BER encoding)</td>
<td>[this document, Section 2]</td>
</tr>
<tr>
<td>TBD110</td>
<td>byte string or array or map</td>
<td>relative object identifier (BER encoding); SDNV [RFC6256] sequence</td>
<td>[this document, Section 2]</td>
</tr>
<tr>
<td>TBD112</td>
<td>byte string or array or map</td>
<td>object identifier (BER encoding), relative to 1.3.6.1.4.1</td>
<td>[this document, Section 2]</td>
</tr>
</tbody>
</table>

7.2. CDDL Control Operators

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>.sdnv</td>
<td>[this document, Section 5]</td>
</tr>
<tr>
<td>.sdnvseq</td>
<td>[this document, Section 5]</td>
</tr>
<tr>
<td>.oid</td>
<td>[this document, Section 5]</td>
</tr>
</tbody>
</table>

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.

An attacker might trick an application into using a byte string inside a tag-factored data item, where the byte string is not actually intended to fall under one of the tags defined here. This may cause the application to emit data with semantics different from what was intended. Applications therefore need to be restrictive with respect to what data items they apply tag factoring to.

9. References

9.1. Normative References

[IANA.cbor-tags]

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


9.2. Informative References

[IANA.enterprise-numbers]
IANA, "Enterprise Numbers",

[OID-INFO] Orange SA, "OID Repository", 2016,

[PCRE] Ho, A., "PCRE - Perl Compatible Regular Expressions",

"Definition of Managed Objects for IPv6 over Low-Power
Wireless Personal Area Networks (6LoWPANs)", RFC 7388,
DOI 10.17487/RFC7388, October 2014,

[X.672] International Telecommunications Union, "Information
technology Open systems interconnection Object
identifier resolution system", ITU-T Recommendation X.672,
Appendix A. Change Log

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

A.1. Changes from -06 to -07

* Various editorial changes prompted by IESG feedback; clarify the usage of "SDNV" in this document (no leading zeros).

* Add security consideration about tag-factoring.

* Make TBD112, where applicable, the preferred serialization (and thus the required deterministic encoding) over TBD111.

A.2. Changes from -05 to -06

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

A.3. Changes from -04 to -05

* Update acknowledgements, contributor list, and author list

A.4. 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.5. Changes from -02 to -03

* Add tag TBD112 for PEN-relative OIDs

* Add suggested CDDL typenames; reference RFC8610

A.6. Changes from -01 to -02

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

A.7. Changes from -00 to -01

Clean up OID tag factoring.
A.8. Changes from -07 (bormann) to -00 (ietf)

   Resubmitted as WG draft after adoption.

A.9. 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.10. Changes from -05 to -06

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

A.11. Changes from -04 to -05

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

A.12. 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.13. 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. Rob Wilton’s IESG review prompted us to provide preferred serialization considerations.

Contributors

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

Email: dev+ietf@seantek.com

Author’s Address

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

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

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 25 July 2021.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal
license-info) in effect on the date of publication of this document.
Please review these documents carefully, as they describe your rights
and restrictions with respect to this document. Code Components
extracted from this document must include Simplified BSD License text
as described in Section 4.e of the Trust Legal Provisions and are
provided without warranty as described in the Simplified BSD License.
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, 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 transferred 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


8.2. Informative References


Contributors

Author’s Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca
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

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 10 August 2021.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
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:

\[ \text{TBD2}([ 24, \text{h'C0000201'}]) \]

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


Author’s Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca