

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

C. Bormann
Universität Bremen TZI
26 December 2021

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

Abstract

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

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

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on 29 June 2022.

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

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

2. Base language features

2.1. Cuts

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

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

3. Literal syntax

3.1. Tag-oriented Literals

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

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

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

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

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 `.regex ...`, which with the proposal in Section 4.1 then raises the question of how to indicate the regular expression flavor.

3.3. Clarifications

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

3.3.1. Err6527

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

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

This allows almost any non-C0 character to be escaped by a backslash, but critically misses out on the `\uXXXX` and `\uHHHH\uLLLL` forms that JSON allows to specify characters in hex. Both can be solved by updating the SESC production to:

```
SESC = "\" ( %x22 / "/" / "\" / ; \" \/ \\
             %x62 / %x66 / %x6E / %x72 / %x74 / ; \b \f \n \r \t
             (%x75 hexchar) ) ; \u
hexchar = non-surrogate / (high-surrogate "\" %x75 low-surrogate)
non-surrogate = ((DIGIT / "A"/"B"/"C" / "E"/"F") 3HEXDIG) /
                ("D" %x30-37 2HEXDIG )
high-surrogate = "D" ("8"/"9"/"A"/"B") 2HEXDIG
low-surrogate = "D" ("C"/"D"/"E"/"F") 2HEXDIG
```

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

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

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

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

3.3.2. Err6543

The ABNF used in [RFC8610] for the content of byte string literals lumps together byte strings notated as text with byte strings notated in base16 (hex) or base64 (but see also updated BCHAR production above):

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

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

```
bytes = %x27 *BCHAR %x27
      / bsqual %x27 *QCHAR %x27
BCHAR = %x20-26 / %x28-5B / %x5D-10FFFD / SESC / CRLF
QCHAR = DIGIT / ALPHA / "+" / "/" / "-" / "_" / "=" / WS
```

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

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

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

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

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

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

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

4. Controls

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

4.1. Control operator .pcre

There are many variants of regular expression languages. Section 3.8.3 of [RFC8610] defines the .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, except for a wider range of regexps).

4.2. Endianness in .bits

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

4.3. .bitfield control

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

```
Field = uint .bitfield Fieldbits
```

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

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

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

5. Co-occurrence Constraints

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

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

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

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

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

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.

6.3. Potential examples

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

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

6.3.1. How name spaces might look like

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

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

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

6.3.2. Explicitly interacting with namespaces

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

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

Besides an implicit import such as

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

there also could be an explicit import syntax:

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

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

6.3.3. Document references

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

6.3.4. Add retroactive exporting to RFCs

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

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

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

6.3.5. Operations

* "export":

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

```
  'oid'  'RFC9090.oid'
```

2. make that namespace available to other specs

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

1. use as is: RFC9090.oid

2. unprefix: oid

Example: prelude processing -- include+unprefix from Appendix D of RFC8610.

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

6.3.6. To be discussed

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

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

Updates, revisions, versions, semver:

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

7. Alternative Representations

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

```
cddltool = ["cddl", +rule]
rule = ["=" / "/" = " / "/" =, namep, type]
namep = ["name", id] / ["gen", id, +id]
id = text .regexp "[A-Za-z@_$_]([[-.])*[A-Za-z0-9@_$_])*"
op = ["..", "..." /
      text .regexp "\\.[A-Za-z@_$_]([[-.])*[A-Za-z0-9@_$_])*"
namea = ["name", id] / ["gen", id, +type]
type = value / namea / ["op", op, type, type] /
      ["map", group] / ["ary", group] / ["tcho", 2*type] /
      ["unwrap", namea] / ["enum", group / namea] /
      ["prim", ?(0..7, ?uint)]
group = ["mem", null/type, type] /
      ["rep", uint, uint/false, group] /
      ["seq", 2*group] / ["gcho", 2*group]
value = ["number"/"text"/"bytes", text]
```

8. IANA Considerations

This document makes no requests of IANA.

9. Security considerations

The security considerations of [RFC8610] apply.

10. References

10.1. Normative References

- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <<https://www.rfc-editor.org/info/rfc9165>>.

10.2. Informative References

- [cddl] "CDDL conversion utilities", n.d., <<https://github.com/cabo/cddl>>.
- [Err6526] "Errata Report 6526", RFC 8610, <<https://www.rfc-editor.org/errata/eid6526>>.
- [Err6527] "Errata Report 6527", RFC 8610, <<https://www.rfc-editor.org/errata/eid6527>>.
- [Err6543] "Errata Report 6543", RFC 8610, <<https://www.rfc-editor.org/errata/eid6543>>.
- [I-D.bormann-cbor-edn-literals]
Bormann, C., "Application-Oriented Literals in CBOR Extended Diagnostic Notation", Work in Progress, Internet-Draft, draft-bormann-cbor-edn-literals-00, 6 October 2021, <<https://www.ietf.org/archive/id/draft-bormann-cbor-edn-literals-00.txt>>.
- [I-D.draft-bormann-jsonpath-iregexp]
Bormann, C., "I-Regexp: An Interoperable Regexp Format", Work in Progress, Internet-Draft, draft-bormann-jsonpath-iregexp-01, 13 November 2021, <<https://www.ietf.org/archive/id/draft-bormann-jsonpath-iregexp-01.txt>>.
- [I-D.handrews-relative-json-pointer]
Luff, G. and H. Andrews, "Relative JSON Pointers", Work in Progress, Internet-Draft, draft-handrews-relative-json-pointer-02, 18 September 2019, <<https://www.ietf.org/archive/id/draft-handrews-relative-json-pointer-02.txt>>.

- [I-D.ietf-core-coral] Amsüss, C. and T. Fossati, "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-04, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-coral-04.txt>>.
- [jsonpath] "jsonpath online evaluator", n.d., <<https://jsonpath.com>>.
- [PCRE] "Perl-compatible Regular Expressions (revised API: PCRE2)", n.d., <<http://pcre.org/current/doc/html/>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, DOI 10.17487/RFC6256, May 2011, <<https://www.rfc-editor.org/info/rfc6256>>.
- [RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <<https://www.rfc-editor.org/info/rfc6901>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/info/rfc7405>>.
- [RFC9090] Bormann, C., "Concise Binary Object Representation (CBOR) Tags for Object Identifiers", RFC 9090, DOI 10.17487/RFC9090, July 2021, <<https://www.rfc-editor.org/info/rfc9090>>.
- [XSD2] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, 28 October 2004, <<https://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.

Acknowledgements

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

Author's Address

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

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 3 December 2021

C. Bormann, Ed.
Universität Bremen TZI
B. Moran
Arm Limited
H. Birkholz
Fraunhofer SIT
E. Cormier
1 June 2021

Map-like data in CBOR and CDDL
draft-bormann-cbor-cddl-map-like-data-01

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.

Basic CBOR supports non-ordered maps free of duplicate keys, similar to the way JSON defines JSON objects (RFC 8259). Using the CBOR extension point of tags, tags for a selection of variants of maps and multimaps have been registered, but gaps remain. The present document defines a consolidated set of CBOR tags for map-like data items involving key-value pairs.

The Concise Data Definition Language (CDDL), standardized in RFC 8610, is often used to express CBOR data structure specifications. It provides "control operators" as its main language extension point. The present document defines a number of control operators that enable the description of CBOR data structures that make use of the newly defined tags or that employ the same underlying structures.

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 3 December 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. CBOR tags for map-like data items	3
2.1. Summary	4
2.2. Key/Value Type	5
2.3. Ordering	5
2.4. Key Uniqueness	5
2.5. Data Item	5
2.6. Related Tags (Informative)	6
2.6.1. Tag 259	6
2.6.2. Tag 275	6
2.6.3. Tag TBD279	6
2.6.4. Tag TDB280	7
3. CDDL Support for Map-Like Data Items	7
3.1. Map notation for map-like data items	7
3.2. Uniqueness	8
4. CDDL typenames	9
5. IANA Considerations	10
5.1. Tags	10
5.2. CDDL control operators	10
6. Implementation Status	10
7. Security considerations	10
8. References	10
8.1. Normative References	10
8.2. Informative References	11
Appendix A. Implementation Considerations	11
A.1. Programming Language Containers (Informative)	11
A.1.1. ECMAScript	12
A.1.2. Python	12
A.1.3. C++	13

A.2. CDDL Implementation Considerations	14
Acknowledgements	14
Contributors	14
Authors' Addresses	14

1. Introduction

(See abstract for now.)

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 [RFC8949] and [RFC8610]. In particular, with respect to CDDL control operators, "target" refers to the left hand side operand, and "controller" to the right hand side operand. The terms "array" and "map" (if unadorned) refer to CBOR major type 4 and CBOR major type 5; this is not called out explicitly.

2. CBOR tags for map-like data items

This document defines a consolidated set of CBOR tags for map-like entities involving key-value pairs. These tags encode the following meta-data concerning map-like data items:

- * the homogeneity of the types of the keys, and of the types of the values;
- * whether the order of the key-value pairs carries semantic value ("ordered") or needs to be ignored ("non-ordered");
- * the uniqueness of the keys; and
- * the major type used to encode the key-value pairs.

Note that the term "ordered" as used in this document is distinct from "sorted" -- "ordered" implies that the order in the data item interchanged conveys a semantically relevant ordering, while a property "sorted" can easily be established after interchange (by, simply, sorting), less often needs to be indicated, and is more complex to indicate as it may need details about the sorting.

2.1. Summary

Tag	LSBs	Homogeneous Value	Homogeneous Key	Ordering	Duplicate Keys Allowed	Data Item	Related Tag
128	0000	No	No	Non-Ordered	No	map	259
129	0001	No	No	Non-Ordered	Yes	array	TBD280*
130	0010	No	No	Ordered	No	array	TBD279*
131	0011	No	No	Ordered	Yes	array	
132	0100	No	Yes	Non-Ordered	No	map	275
133	0101	No	Yes	Non-Ordered	Yes	array	
134	0110	No	Yes	Ordered	No	array	
135	0111	No	Yes	Ordered	Yes	array	
136	1000	Yes	Yes	Non-Ordered	No	map	
137	1001	Yes	Yes	Non-Ordered	Yes	array	
138	1010	Yes	Yes	Ordered	No	array	
139	1011	Yes	Yes	Ordered	Yes	array	

Table 1: New CBOR tags defined in this document

*TBD279: https://github.com/Sekenre/cbor-ordered-map-spec/blob/master/CBOR_Ordered_Map.md

*TBD280: <https://github.com/ecorm/cbor-tag-multimap>

[The intention of the present document is to obviate the need for defining TBD279/TBD280.]

Appendix A.1 provides information about constructs in a few programming languages that are related to the tags being defined.

2.2. Key/Value Type

Bits 2 and 3 of the tag provide information on the map's key and value types:

- * 0b00xx Unspecified: There is no specified type for the map's keys and values
- * 0b01xx Homogeneous Key: All keys have the same data type
- * 0b10xx Homogeneous Key/Value: All values have the same data type in addition to all keys having the same data type (the types for keys and values may be distinct).

The semantics for homogeneity shall be the same as for [RFC8746] homogeneous arrays (tag 41). That is, "which CBOR data items constitute elements of the same application type is specific to the application" (Section 3.2 of [RFC8746]).

Maps with arbitrary keys and homogeneous values are considered unusual, so they are left out of this specification so that fewer tag numbers need to be allocated (12 instead of 16).

2.3. Ordering

Bit 1 of the tag represents the map's ordering semantics:

- * 0: The order of key-value pairs is unspecified
- * 1: Key-value pairs are encoded in the same order in which they were inserted

2.4. Key Uniqueness

Bit 0 of the tag represents the uniqueness of the map's keys.

- * 0: Keys are unique within the map
- * 1: Keys may be duplicated (i.e., multimaps)

2.5. Data Item

All these map-like data items could be represented as a tag with an enclosed array of alternating key-value pairs, as in:

```
129(["key1", 1, "key2", 2])
```

However, representing the key-value pairs as a CBOR map for those cases where this is possible enables generic decoders that are oblivious of these tags to represent the data in a more appropriate platform type.

Specifically, the key-value pairs are represented as a map if and only if

- * the ordering is unspecified and
- * the keys are unique;

otherwise, they are represented as an array of alternating keys and values ("flattened alist", see Figure 1).

```
FAList<K, V> = [* (K, V)]
```

Figure 1: CDDL for order-preserving representation of maps

Issue: [MAPREP] discusses alternative representations of (ordered and other) maps. How much of this do we need to address here?

2.6. Related Tags (Informative)

2.6.1. Tag 259

Specification: <https://github.com/shanewholloway/js-cbor-codec/blob/master/docs/CBOR-259-spec--explicit-maps.md>

The above defined tag 128 may be used instead to guide a JavaScript decoder into interpreting a CBOR map as a JavaScript Map instead of an Object.

2.6.2. Tag 275

Specification: <https://github.com/ecorm/cbor-tag-text-key-map>

The above defined tag 132 may be used instead to guide a decoder into interpreting a CBOR map as a JavaScript-like Object having only text string keys. The decoder would have to verify the first key to establish that the map has homogeneous text string keys.

2.6.3. Tag TBD279

Draft specification: https://github.com/Sekenre/cbor-ordered-map-spec/blob/master/CBOR_Ordered_Map.md

The above defined tag 130 may be used instead to encode map-like data items where the order of the key-value pairs is semantically significant.

2.6.4. Tag TDB280

Draft specification: <https://github.com/ecorm/cbor-tag-multimap>

The above defined tag 129 may be used instead to encode a multimap as an array of key-value pairs.

3. CDDL Support for Map-Like Data Items

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 enable the use of group notation (enclosed in a CDDL map) to specify any of the above map-like data structures:

Name	Purpose
.omm	Ordered (Multi-)Map
.nomm	Non-Ordered (Multi-)Map
.unique	Uniqueness requirement

Table 2: New control operators in this document

3.1. Map notation for map-like data items

[needs better examples]

CDDL already can describe both arrays of alternating keys and values and maps (non-ordered and with unique keys). The two control operators ".omm" and ".nomm" introduced in this section enable the use of CDDL map notation for map-like types beyond actual maps, increasing readability and possibly even reusability.

In a simple example that provides an non-ordered collection of zero or more home addresses and zero or more work addresses, each labeled as such, we use traditional map notation to describe that collection:

```
[* (text, any)] .nomm {  
  * home: address  
  * work: address  
  $$more-addresses  
}
```

The ".omm" and ".nomm" control operators convert a group definition enclosed into a CDDL map given as a controller type into an array type given as the target type. The controller type given is unwrapped (Section 3.7 of [RFC8610]) into a group. Keys and values of the entries in that group are then alternately matched as elements in the target array. Note that both target and controller type can contribute to the shaping of the data; declaring the key type as "text" limits what can be added to the "\$\$more-addresses" socket.

".omm" and ".nomm" differ in the semantics of the array type created: ".omm" defines an ordered (multi)map, i.e., the order of the key/value element pairs in the array matters, while ".nomm" defines a non-ordered (multi)map, i.e., data items that present the same set of key/value pairs in different orders are equivalent.

The ability to specify specific ("homogeneous") types is provided by the ability to specify the target type, as in the example above.

Note that there is not strictly a need to define a control operator for building non-ordered maps with non-duplicate keys, as existing CBOR maps already fill this role, however the use of a map type as the target is allowed for symmetry (implying uniqueness of the keys), allowing the following:

```
{* text => any} .nomm {  
  ? home: address  
  ? work: address  
  $$more-addresses  
}
```

3.2. Uniqueness

The ".unique" control annotates the target as requiring uniqueness, within the enclosing container(*), of its value, among the other data items in that enclosing container that are also marked ".unique", under the same label (given as the controller).

E.g.,

```
feature-set = [* feature .unique "set"]
ordered-pairs-with-unique-keys-and-values =
  [* (any .unique "key", any .unique "value") ]
```

defines a "feature-set" as an array of zero or more "feature" values that need to be all different (as they are unique under the label "set"), and "ordered-map-with-unique-keys-and-values" as an array of zero pairs of keys and values, where the keys need to be unique among themselves and the values need to be unique among themselves (the latter example could employ an ".omm" or ".nomm" operator to further restrict what can be in these keys and values).

Discussion: (*) while it is probably not a big problem to define what exactly the "enclosing" container is, it may be useful to actually define a larger scope of the uniqueness. CDDL currently does not have a way to establish and point to such a larger scope; we might define one ad hoc here or leave that for later extension.

4. CDDL typenames

For the use with CDDL [RFC8610], the typenames defined in Figure 2 are recommended unless there is a need for more specific shaping of the data.

```
anymap = { * any => any }

tbd128 = #6.128(anymap)
tbd129 = #6.129([* (any, any)] .nomm anymap)
tbd130 = #6.130([* ((any .unique "mm"), any)] .omm anymap)
tbd131 = #6.131([* (any, any)] .omm anymap)
tbd132<k> = #6.132({ * k => any })
tbd133<k> = #6.133([* (k, any)] .nomm anymap)
tbd134<k> = #6.134([* ((k .unique "mm"), any)] .omm anymap)
tbd135<k> = #6.135([* (k, any)] .omm anymap)
tbd136<k,v> = #6.136({ * k => v })
tbd137<k,v> = #6.137([* (k, v)] .nomm anymap)
tbd139<k,v> = #6.138([* ((k .unique "mm"), v)] .omm anymap)
tbd139<k,v> = #6.139([* (k, v)] .omm anymap)
```

Figure 2: Recommended typenames for CDDL

Issue: fill in better names for tbdnnn

Note that there is no need to call out the uniqueness of the keys explicitly in tbd128, tbd132, or tbd136, as the use of maps as a representation format already provides that key uniqueness.

5. IANA Considerations

5.1. Tags

IANA is requested to allocate the tags of Table 1 in the CBOR tags registry [IANA.cbor-tags], using this document as the specification reference.

The allocations are requested to be assigned from the "specification required" space (24..255). The values in the column labeled "Tag" in Table 1 are suggested as the allocated tag numbers.

5.2. CDDL control operators

This document requests IANA to register the contents of Table 3 into the CDDL Control Operators registry [IANA.cddl]:

=====	=====
Name	Reference
=====	=====
.omm	[RFCthis]
-----	-----
.nomm	[RFCthis]
-----	-----
.unique	[RFCthis]
-----	-----

Table 3: New control operators to be registered

6. Implementation Status

TBD

7. Security considerations

The security considerations of [RFC8610] apply.

8. References

8.1. Normative References

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

- [IANA.cddl] IANA, "Concise Data Definition Language (CDDL)",
<<http://www.iana.org/assignments/cddl>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data
Definition Language (CDDL): A Notational Convention to
Express Concise Binary Object Representation (CBOR) and
JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610,
June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8746] Bormann, C., Ed., "Concise Binary Object Representation
(CBOR) Tags for Typed Arrays", RFC 8746,
DOI 10.17487/RFC8746, February 2020,
<<https://www.rfc-editor.org/info/rfc8746>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object
Representation (CBOR)", STD 94, RFC 8949,
DOI 10.17487/RFC8949, December 2020,
<<https://www.rfc-editor.org/info/rfc8949>>.

8.2. Informative References

- [MAPREP] Bormann, C., "Re: [Cbor] "ordered hash"", cbor@ietf.org
mailing list message, 30 July 2020,
<<https://mailarchive.ietf.org/arch/msg/cbor/5MuDSyPivZ7JfPhsfwCaW2usFHQ>>.

Appendix A. Implementation Considerations

This non-normative appendix provides information about the use on
implementations of the tags and control operators defined.

A.1. Programming Language Containers (Informative)

The following subsections describe how the tags in this document
relate to various programming language containers. Containers that
are not part of the programming language or its standard libraries
are not considered here.

The `_Encoding Tag_` column in the following tables provide the recommended tag that best represents the given container type. For example, it's possible to use tag 132 for encoding an ECMAScript "Map" if all keys happen to be of the same type, however tag 128 is more general and applies to any "Map". When encoding an ECMAScript "Object", tag 128 would be technically correct but is too general; tag 132 best presents the fact that an "Object" has text keys only.

The `_Decodable Tags_` column in the following tables, are for data items can be decoded into the destination container without having to inspect the following:

- * the uniqueness of the keys,
- * the ordering of the keys, and,
- * the data types of *every* keys/value pair.

It may however be necessary to inspect the data types of the *first* key-value pair in the case of tags representing homogeneous keys/values.

A.1.1. ECMAScript

Container	Encoding Tag	Decodable Tags
Object	132	132, 136
Map	128	128, 132, 136
"Array" of pairs	131	All

Table 4

A.1.2. Python

Container	Encoding Tag	Decodable Tags
TypedDict	136	136
namedtuple	132	132, 136
dict	128	128, 132, 136
OrderedDict	130	130, 134, 138

"list" of 2-tuples	131	All
--------------------	-----	-----

Table 5

A.1.3. C++

Container(s)	Encoding Tag	Decodable Tags
Map<K, T>	136	136
Map<K, D>	132	132, 136
Map<D, D>	128	128, 132, 136
MultiMap<K, T>	137	137
MultiMap<K, D>	133	133
MultiMap<D, D>	129	128, 129
Sequence<Pair<K, T>>	139	[136, 139]
Sequence<Pair<K, D>>	135	[132, 139]
Sequence<Pair<D, D>>	131	All

Table 6

Legend:

- * "K": Static key type
- * "T": Static value type
- * "D": Suitable dynamic type, such as "std::any" or "std::variant"
- * "Map": "std::map" or "std::unordered_map"
- * "MultiMap": "std::multimap" or "std::unordered_multimap"
- * "Sequence": Sequence container that maintains order (e.g. "std::vector")

- * "Pair": Object containing a key and a value, such as "std::pair", or "std::tuple".

Note that a C++ "std::map" stores its key-value pairs in a sorted fashion, and does *not* preserve insertion order in the same manner as Python's "OrderedDict".

A.2. CDDL Implementation Considerations

TBD

Acknowledgements

The CBOR tags defined in this document were developed by Emile Cormier under the sponsorship of Duc Luong, based on discussions with Kio Smallwood and Joe Hildebrand. The CDDL control operators defined in this document were developed by Carsten Bormann, Brendan Moran, and Henk Birkholz.

Contributors

Kio Smallwood

Joe Hildebrand

Authors' Addresses

Carsten Bormann (editor)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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

Brendan Moran
Arm Limited

Email: Brendan.Moran@arm.com

Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

Emile Cormier

Email: emile.cormier.jr@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 27 August 2022

C. Bormann
Universität Bremen TZI
23 February 2022

Notable CBOR Tags
draft-bormann-cbor-notable-tags-06

Abstract

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

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.

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 August 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
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	7
5. Protocols	8
5.1. DOTS	8
5.2. RAINS	8
6. Datatypes	8
6.1. Advanced arithmetic	9

6.2. Variants of undefined	11
6.3. Typed and Homogeneous Arrays	11
7. Domain-Specific	12
7.1. Extended Time Formats	13
8. Platform-oriented	14
8.1. Perl	15
8.2. JSON	15
8.3. Weird text encodings	16
9. Application-specific	16
9.1. Enumerated Alternative Data Items	17
9.1.1. Semantics	18
9.1.2. Rationale	19
9.1.3. Examples	20
10. Implementation aids	21
10.1. Invalid Tag	21
11. IANA Considerations	21
12. Security Considerations	22
13. References	22
13.1. Normative References	22
13.2. Informative References	23
Acknowledgements	26
Contributors	26
Author's Address	26

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 [STD94] apply. Specifically: The term "byte" is used in its now customary sense as a synonym for "octet"; "byte strings" are CBOR data items carrying a sequence of zero or more (binary) bytes, while "text strings" are CBOR data items carrying a sequence of zero or more Unicode code points, encoded in UTF-8 [STD63]. 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: 2⁶⁴) 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.

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

34	UTF-8 string	base64	2.4.4.3
35	UTF-8 string	Regular expression	2.4.4.3
36	UTF-8 string	MIME message	2.4.4.3
55799	multiple	Self-describe CBOR	2.4.5

Table 1: Tag numbers defined in RFC 7049

2.1. Tags Related to Those Defined in RFC 7049

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

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

Appendix G.3 of [STD94] 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 as "ECMA262" 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].

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

Table 2: Tag numbers defined in RFC 8152, COSE

3.2. RFC 8392 (CWT)

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

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

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

4. CBOR-based Representation Formats

Representation formats can be built on top of CBOR.

4.1. YANG-CBOR

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

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

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

Table 4: Tag number defined for YANG-CBOR

5. Protocols

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

5.1. DOTS

DDoS Open Threat Signaling (DOTS) defines tag number 271 for the DOTS signal channel object in [RFC9132].

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 for the URL is given in Table 5.

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

Table 5: Tags for advanced arithmetic

CBOR's basic generic data model (Section 2 of [STD94]) 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 [STD94]) 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])
rational_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.

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; explain how tags 52 and 54 essentially obsolete 260/261.)

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

Table 7

7.1. Extended Time Formats

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

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

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 [STD94].)

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.)

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

Table 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>).

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

Table 10: Tag numbers for UTF-8 variants

9. Application-specific

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

Tag number	Tag Author content	Short Description	Reference
39	multiple Lucas Clemente	Identifier	[https://github.com/lucas-clemente/cbor- specs/blob/master/id.md]
42	byte Volker string Mische	IPLD content identifier	[https://github.com/ipld/cid-cbor/
103	array Danilo Vidovic	Geographic Coordinates	[https://github.com/allthingstalk/cbor/b lob/master/CBOR-Tag103-Geographic-Coordinates.md]
104	multiple	Geographic Coordinate Reference System WKT or EPSG number	[I-D.clarke-cbor-crs]
120	multiple Danilo Vidovic	Internet of Things Data Point	[https://github.com/allthingstalk/cbor/b lob/master/CBOR-Tag120-Internet-of-Things-Data-Poin ts.md]
258	array Alfredo Di Napoli	Mathematical finite set	[https://github.com/input-output-hk/cbor- spec/blob/master/CBOR_SETS.md]
259	map Shane Holloway	Map datatype with key-value operations (e.g. .get ()/.set ()/.delete ())	[https://github.com/shanewholloway/js-cb or-codec/blob/master/docs/CBOR-259-spec--ex plicit-maps.md]

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

Table 11

9.1. Enumerated Alternative Data Items

(Original Text for this section was contributed by Duncan Coutts and Michael Peyton Jones; all errors are the author's.)

A set of CBOR tag numbers has been allocated (to do, Section 11) for encoding data composed of enumerated alternatives:

Tags	Data Item	Meaning
121..127	any	alternatives 0..6, 1+1 encoding
1280..1400	any	alternatives 7..127, 1+2 encoding
101	array [uint, any]	alternatives as given by the uint + 128

Table 12: Tags for Enumerated Alternative Data Items

The tags defined in this section are for encoding data that can be in one of a number of different enumerated forms.

For example data representing the result of some action might be either a failure with some failure detail, or a success with some result. In this example there are two cases, the failure case and the success case, and we can enumerate them as 0 and 1.

In general the number of alternatives, and what data is expected in each alternative case is entirely application dependent.

The tags defined in this specification allow the encoding of any number of alternatives, but provide compact encoding for the common cases of low numbers of alternatives:

- * Alternatives 0..6 can be encoded in 2 bytes;
- * Alternatives 7..127 can be encoded in 3 bytes;
- * Alternatives 128+ can be encoded in 3-12 bytes.

There are no special considerations for deterministic encoding Section 4.2 of [STD94]: The case numbers covered by each tag do not overlap; particularly, tag 101 encoding starts where the more compact special encodings for 0..6 and 7..127 end.

9.1.1. Semantics

The value consists of a case number and a case body. The case number is an unsigned integer that indicates which case out of the set of alternatives is used. The case body is any CBOR data value.

In a setting where the application uses a schema (formally or informally), then there will be an appropriate sub-schema for each case in the set of alternatives. The representation of the case body should comply with the schema corresponding to the case number used.

To continue the example above about representing failure or success, suppose that the failure detail consists of an integer code and a string, and suppose that the successful result is a byte string. A failure value will use case 0 and the case body will be a CBOR list containing an integer and a text string. Alternatively, a success value will use case 1 and the body will be a single CBOR byte string.

Decoders that enforce a schema must check the case number is within the range of cases allowed, and that the case body follows the schema for the supplied case number. Generic decoders should allow any case number and any CBOR data value for the case body.

9.1.2. Rationale

CBOR has direct support for combinations of multiple values but not for alternatives of multiple values. Combinations are expressed in CBOR using lists or maps.

Most programming languages have a notion of data consisting of combinations of data values, often called records or objects. Many programming languages also have a notion of data consisting of multiple alternative data values. For example C has unions, and other languages have "tagged" unions (where it is always clear which alternative is in use).

Crucially for this set of tags, the set of alternatives must be closed and ordered. This allows encoding using an unsigned number to distinguish each case.

Note that this does not correspond to the notion in some programming languages of classes and subclasses since in that context the set of alternatives is open and unordered. Alternatives of this kind are well-supported by tag 27 "Serialized language-independent object with type name and constructor arguments".

In functional programming languages, the primary way of forming new data types is to enumerate a set of alternatives (each of which may be a record). Such forms of data are also supported in hybrid functional languages or languages with functional features.

Thus, in some applications, it is very common to have data making use of alternatives, and it is worth finding a compact encoding, at least for the common cases. Just as most records are small, most alternatives are also small.

In this specification we reserve 7 values in the 2-byte part of the available tag encoding space for alternatives 0..6 which are by far the most common. We reserve a range of 121 values in the 3-bytes tag encoding space. To cover the general case we use an encoding using a pair consisting of an unsigned integer and the case body, the first 24 of which also result in a 3-byte encoding.

9.1.3. Examples

To elaborate on the example from the introduction, we have a "result" that is a failure or success, where:

- * the failure detail consists of an integer code and a string;
- * the successful result is a byte string.

This corresponds to the following schema, in CDDL notation:

```
result = #6.121([int, text])  
        / #6.122(bytes)
```

Example values:

```
121([3, "the printer is on fire"])
```

```
122(h'ff00')
```

As a second example, here is one based on a data type defined within the Haskell programming language, representing a simple expression tree.

```
-- A data type representing simple arithmetic expressions
```

```
data Expr = Lit Int -- integer literal  
          | Add Expr Expr -- addition  
          | Sub Expr Expr -- subtraction  
          | Neg Expr -- unary negation  
          | Mul Expr Expr -- multiplication  
          | Div Expr Expr -- integer division
```

In CDDL notation, and using the tags in this specification, such data could be encoded using this schema:

; A data type representing simple arithmetic expressions

```
expr = 121(int)           ; integer literal
      / 122([expr, expr]) ; addition
      / 123([expr, expr]) ; subtraction
      / 124(expr)         ; unary negation
      / 125([expr, expr]) ; multiplication
      / 126([expr, expr]) ; integer division
```

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 13 from the FCFS space, with the present document as the specification reference. IANA has allocated the fourth tag from the Specification Required space, with the present document as the specification reference.

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

Table 13: Values for Tags

In addition, IANA is requested to allocate the tags from Table 12, with a reference to the present document.

12. Security Considerations

The security considerations of [STD94] apply; the tags discussed here may also have specific security considerations that are mentioned in their specific sections above.

13. References

13.1. Normative References

[I-D.ietf-core-yang-cbor]

Veillette, M., Petrov, I., Pelov, A., Bormann, C., and M. Richardson, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-18, 19 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-yang-cbor-18.txt>>.

[IANA.cbor-tags]

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

- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8746] Bormann, C., Ed., "Concise Binary Object Representation (CBOR) Tags for Typed Arrays", RFC 8746, DOI 10.17487/RFC8746, February 2020, <<https://www.rfc-editor.org/info/rfc8746>>.
- [RFC9132] Boucadair, M., Ed., Shallow, J., and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 9132, DOI 10.17487/RFC9132, September 2021, <<https://www.rfc-editor.org/info/rfc9132>>.
- [STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

13.2. Informative References

- [C] International Organization for Standardization, "Information technology - Programming languages - C", ISO/IEC 9899:2018, June 2018, <<https://www.iso.org/standard/74528.html>>.
- [Cplusplus20] International Organization for Standardization, "Programming languages - C++", ISO/IEC ISO/IEC JTC1 SC22 WG21 N 4860, March 2020, <<https://isocpp.org/files/papers/N4860.pdf>>.

[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, <<https://www.ietf.org/archive/id/draft-clarke-cbor-crs-02.txt>>.

[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, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-algs-12.txt>>.

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

Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

[I-D.trammell-rains-protocol]

Trammell, B. and C. Fehlmann, "RAINS (Another Internet Naming Service) Protocol Specification", Work in Progress, Internet-Draft, draft-trammell-rains-protocol-05, 29 January 2019, <<https://www.ietf.org/archive/id/draft-trammell-rains-protocol-05.txt>>.

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2019, DOI 10.1109/IEEESTD.2019.8766229, <<https://ieeexplore.ieee.org/document/8766229>>.

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

- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <<https://www.rfc-editor.org/info/rfc7322>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [RFC8943] Jones, M., Nadalin, A., and J. Richter, "Concise Binary Object Representation (CBOR) Tags for Date", RFC 8943, DOI 10.17487/RFC8943, November 2020, <<https://www.rfc-editor.org/info/rfc8943>>.

[STD63] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.

Acknowledgements

(Many, TBD)

Contributors

Peter Occil
Email: poccill14 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.

Duncan Coutts
Email: duncan@well-typed.com

Michael Peyton Jones
Email: me@michaelpj.com

Jane Doe
To do

Further contributors will be listed here as text is added.

Plase 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 April 2022

C. Bormann
Universität Bremen TZI
22 October 2021

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

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. Computed Literals	3
2.1. Numeric Addition	4
2.2. String Concatenation	4
2.3. String Concatenation with Dedenting	5
3. Embedded ABNF	6
4. Features	8
5. IANA Considerations	10
6. Implementation Status	11
7. Security considerations	11
8. References	12
8.1. Normative References	12
8.2. Informative References	12
Acknowledgements	13
Author's Address	13

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:

Name	Purpose
.plus	Numeric addition
.cat	String Concatenation
.det	String Concatenation, pre-dedenting
.abnf	ABNF in CDDL (text strings)
.abnfb	ABNF in CDDL (byte strings)
.feature	Indicate name of feature used (extension point)

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

Figure 2: Example: concatenation of text and byte string

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

2.3. String Concatenation with Dedenting

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

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

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

Figure 3: Example: dedenting concatenation

The control operator `.det` works like `.cat`, except that 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.

```
oid = bytes .abnfb 'oid
oid = 1*arc
roid = *arc
arc = [nlsb] %x00-7f
nlsb = %x81-ff *%x80-ff
,
```

Figure 4: Dedenting example: result of first .det

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].

```

; for RFC 8943
Tag1004 = #6.1004(text .abnf full-date)
; for RFC 8949
Tag0 = #6.0(text .abnf date-time)

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

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

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

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

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

Figure 5: Example: employing RFC 3339 ABNF for defining CBOR Tags

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]:

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

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

- [IANA.cddl] IANA, "Concise Data Definition Language (CDDL)", <<https://www.iana.org/assignments/cddl>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<https://www.rfc-editor.org/info/rfc7405>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

8.2. Informative References

- [CDDL-RS] Weiss, A., "cddl-rs", n.d., <<https://github.com/anweiss/cddl>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <<https://www.rfc-editor.org/info/rfc8428>>.

- [RFC8943] Jones, M., Nadalin, A., and J. Richter, "Concise Binary Object Representation (CBOR) Tags for Date", RFC 8943, DOI 10.17487/RFC8943, November 2020, <<https://www.rfc-editor.org/info/rfc8943>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

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

CBOR Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 November 2022

M. Richardson
Sandelman Software Works
C. Bormann
Universität Bremen TZI
5 May 2022

On storing CBOR encoded items on stable storage
draft-ietf-cbor-file-magic-12

Abstract

This document defines a stored ("file") format for CBOR data items that is friendly to common file type recognition systems such as the Unix file(1) command.

About This Document

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

Status information for this document may be found at
<https://datatracker.ietf.org/doc/draft-ietf-cbor-file-magic/>.

Discussion of this document takes place on the cbor Working Group mailing list (<mailto:cbor@ietf.org>), which is archived at
<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>.

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 6 November 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	4
1.2. Requirements for a Magic Number	5
2. Protocol	5
2.1. The CBOR Protocol Specific Tag	5
2.2. Enveloping Method: CBOR Tag Wrapped	6
2.2.1. Example	7
2.3. Enveloping Method: Labeled CBOR Sequence	7
2.3.1. Example	8
3. Security Considerations	9
4. IANA Considerations	9
4.1. Labeled CBOR Sequence Tag	10
4.2. CBOR-Labeled Non-CBOR Data Tag	10
4.3. CBOR Tags for CoAP Content-Format Numbers	11
5. References	11
5.1. Normative References	12
5.2. Informative References	12
Appendix A. Advice to Protocol Designer	14
A.1. Is the on-wire format new?	15
A.2. Can many items be trivially concatenated?	15
A.3. Are there tags at the start?	16
Appendix B. CBOR Tags for CoAP Content Formats	16
B.1. Content-Format Tag Examples	18
Appendix C. Example from Openswan	18
Appendix D. Using CBOR Labels for non-CBOR data	19
D.1. Content-Format Tag Examples	20
Appendix E. Changelog	20
Acknowledgements	20
Contributors	21
Authors' Addresses	21

1. Introduction

Since very early in computing, operating systems have sought ways to mark which files could be processed by which programs. In Unix, everything is a stream of bytes; identifying the contents of a stream of bytes became a heuristic activity.

For instance, the Unix `file(1)` command, which has existed since 1973 [file], has for decades been able to identify many file formats 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, in classical macOS, a resource fork was maintained separately from the file data that included file type information; this way, the OS ideally never needed to know anything about the file data contents to determine the media type.

Many other systems do this by file extensions. Many common web servers derive the media-type information from file extensions.

Having a media type associated with the file contents can avoid some of the brittleness of this approach. When files become disconnected from their type information, such as when attempting to do forensics on a damaged system, then being able to identify the type of information that is stored in file can become very important.

A common way to identify the type of a file from its contents is to place a "magic number" at the start of the file contents [MAGIC]. It is noted that in the media type registration template [RFC6838], a magic number is asked for, if available, as is a file extension.

A challenge for the `file(1)` command is often that it can be confused by the encoding vs. the content. For instance, an Android "apk" (as 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, and may also be identified as a ZIP file.

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. As a CBOR format may use a single CBOR data item or a CBOR sequence of data items [RFC8742], two possible methods of enveloping data are presented; a CBOR Protocol designer will specify one. (A CBOR Protocol is a specification which uses CBOR as its encoding.)

This document also gives advice to designers of CBOR Protocols on choosing one of these mechanisms for identifying their contents. This advice is informative.

A third method is also proposed by which this CBOR format prepended tag is used to identify non-CBOR files. This third method has been placed in Appendix D because it is not about identifying media types containing CBOR-encoded data items. This includes a simple way to derive a magic number to content-formats as defined by [RFC7252], even if not in CBOR form.

Examples of CBOR Protocols currently under development include Concise Software Identification Tags (CoSWID, [I-D.ietf-sacm-coswid]) and Entity Attestation Tokens (EAT, [I-D.ietf-rats-eat]). COSE itself [RFC8152] is considered infrastructure. The encoding of public keys in CBOR as described in [I-D.ietf-cose-cbor-encoded-cert] as `_C509_` would benefit from being an identified CBOR Protocol.

A major inspiration for this document is observing the disarray in certain ASN.1 based systems where most files are PEM encoded; these are then all identified by the extension "pem", confusing public keys, private keys, certificate requests, and S/MIME content.

While the envelopes defined in this specification 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. However, 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 can be used to signal that a CBOR Protocol is being used as opposed to a legacy scheme.

1.1. Terminology

Byte is a synonym for octet. The term "byte string" refers to the data item defined in [STD94].

The term "file" is understood to stand in a general way for a stored representation that is somewhat detached from the original context of usage of that representation; its usage in this document encompasses similar units of storage that may have different identification schemes such as partitions or media blocks.

The term "diagnostic notation" refers to the human-readable notation for CBOR data items defined in Section 8 of [STD94] and Appendix G of [RFC8610].

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

The function TN(ct) is defined in Appendix B.

1.2. Requirements for a Magic Number

A magic number is ideally a fingerprint that is unique to a specific 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

This Section presents two enveloping methods. Both use CBOR Tags in a way that results in a deterministic first 8 to 12 bytes. Which one is to be used is up to the CBOR Protocol designer to determine; see Appendix A for some guidance.

2.1. The CBOR Protocol Specific Tag

In both enveloping methods, CBOR Protocol designers need to obtain a CBOR tag for each kind of object that they might store in files. 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 a simple interaction (e.g., via web or email) with IANA, having filled in the small template provided in Section 9.2 of [STD94]. In the template, it is suggested to include a reference to this specification (RFC XXXX) alongside the Description of semantics.

```
// (Note to RFC Editor: Please replace all occurrences of "RFC XXXX"  
// with the RFC number of the present specification and remove this  
// note.)
```

Allocation of the CBOR tag needs to be initiated by the designer of the CBOR Protocol, who can provide a proposed tag number. In order to be in the four-byte range, and so that there are no leading zero bytes in the four-byte encoding of the tag number, the value needs to be in the range 0x01000000 (decimal 16777216) to 0xFFFFFFFF (decimal 4294967295) inclusive. It is further suggested to avoid values that have an embedded zero byte in the four bytes of their binary representation (such as 0x12003456), as these may confuse implementations that treat the magic number as a C string.

The use of a sequence of four US-ASCII [RFC20] codes which are mnemonic to the protocol is encouraged, but not required (there may be reasons to encode other information into the tag; see Appendix B for an example). For instance, Appendix C uses "OPSN" which translates to the tag number 1330664270 registered for it.

For CBOR data items that form 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 proactively been allocated in Section 4.3 (see Appendix B 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. This data item is enveloped into two nested tags:

The outer tag is a Self-described CBOR tag, 55799, as described in Section 3.4.6 of [STD94].

The tag content of the outer tag is a second CBOR tag whose tag number has been allocated to describe the specific Protocol involved, as discussed in Section 2.1. The tag content of this inner tag is the single CBOR data item.

This method wraps the CBOR data item as CBOR tags usually do. Applications that need to send the stored CBOR data item across a constrained network may wish to remove the two tags if the type is understood from the protocol context, e.g., from a CoAP Content-Format Option (Section 5.10.3 of [RFC7252]). A CBOR Protocol specification may therefore pick the specific cases where the CBOR Tag Wrapped enveloping method is to be used. For instance, it might specify its use for storing the representation in a local file or for Web access, but not within protocol messages that already provide the necessary context.

2.2.1. Example

To construct an example without registering a new tag, we use the Content-Format number registered in [RFC8428] for application/senml+cbor (as per Registry Content-Formats of [IANA.core-parameters]), the number 112.

Using the technique described in Appendix B, this translates into the tag TN(112) = 1668546929.

With this tag, the SenML-CBOR pack [{0: "current", 6: 3, 2: 1.5}] would be enveloped as (in diagnostic notation):

```
55799(1668546929([{0: "current", 6: 3, 2: 1.5}]))
```

Or in hex:

d9 d9f7	# tag(55799)
da 63740171	# tag(1668546929)
81	# array(1)
a3	# map(3)
00	# unsigned(0)
67	# text(7)
63757272656e74	# "current"
06	# unsigned(6)
03	# unsigned(3)
02	# unsigned(2)
f9 3e00	# primitive(15872)

At the representation level, the unique fingerprint for application/senml+cbor is composed of the 8 bytes d9d9f7da63740171 hex, after which the unadorned CBOR data (81... for the SenML data) is appended.

2.3. Enveloping Method: Labeled CBOR Sequence

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

This method prepends a newly constructed, separate data item to the CBOR Sequence, the `_label_`.

The label is a nesting of two tags, similar to but distinct from the CBOR Tag Wrapped methods, with an inner tag content of a constant byte string. The total length of the label is 12 bytes.

1. The outer tag is the self-described CBOR Sequence tag, 55800.

2. The inner tag is a CBOR tag, from the First Come First Served space, that uniquely identifies the CBOR Protocol. As with CBOR Tag Wrapped, the use of a four-byte tag is encouraged that encodes without zero bytes.
3. The tag content is a three byte CBOR byte string containing 0x42_4f_52 ('BOR' in diagnostic notation).

The outer tag in the label identifies the file as being a CBOR Sequence, and does so with all the desirable properties explained in Section 3.4.6 of [STD94]. Specifically, it does not appear to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The inner tag in the label identifies which CBOR Protocol is used, as described above.

The inner tag content is a constant byte string which is represented as 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three-byte string 0x42_4f_52 ('BOR' in diagnostic notation).

The actual CBOR Protocol data then follow 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 inner nested tag needs to tag something.

2.3.1. Example

To construct an example without registering a new tag, we use the Content-Format number registered in [RFC9177] for application/missing-blocks+cbor-seq (as per Registry Content-Formats of [IANA.core-parameters]), the number 272.

Using the technique described in Appendix B, this translates into the tag TN(272) = 1668547090.

This is a somewhat contrived example, as this is not a media type that is likely to be committed to storage. Nonetheless, with this tag, missing blocks list 0, 8, 15 would be enveloped as (in diagnostic notation):

```
55800(1668547090('BOR')),  
0,  
8,  
15
```

Or in hex:

```
# CBOR sequence with 4 elements  
d9 d9f8      # tag(55800)  
  da 63740212 # tag(1668547090)  
    43        # bytes(3)  
      424f52  # "BOR"  
00 # unsigned(0)  
08 # unsigned(8)  
0f # unsigned(15)
```

At the representation level, the unique fingerprint for application/missing-blocks+cbor-seq is composed of the 8 bytes d9d9f8da63740212 hex, after which the unadorned CBOR sequence (00... for the missing block list given) is appended.

3. Security Considerations

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

The most obvious is that it may allow malware to identify interesting stored objects, and then exfiltrate or corrupt them.

Protective applications (that check data) cannot rely on the applications they try to protect (that use the data) to make exactly the same decisions in recognizing file formats. (This is an instance of a check vs. use issue.) For example, end-point assessment technologies should not solely rely on the labeling approaches described in this document to decide whether to inspect a given file. Similarly, depending on operating systems configurations and related properties of the execution environment the labeling might influence the default application used to process a file in a way that may not be predicted by a protective application.

4. IANA Considerations

These IANA considerations are entirely about CBOR Tags, in the registry CBOR Tags of [IANA.cbor-tags].

Section 4.1 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 C). Section 4.3 allocates a CBOR tag for each actual or potential CoAP Content-Format number (examples are in Appendix B).

4.1. Labeled CBOR Sequence Tag

IANA has allocated tag 55800 as the tag for the Labeled CBOR Sequence Enveloping Method from the CBOR Tags Registry. IANA is asked to update this tag registration to point to this document.

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 [STD94]).

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 [STD94] 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

4.2. CBOR-Labeled Non-CBOR Data Tag

IANA is requested to allocate tag 55801 as the tag for the CBOR-Labeled Non-CBOR Data Enveloping Method (Appendix D) from the CBOR Tags Registry. IANA is asked to update this tag registration to point to this document.

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 [STD94]).

The hexadecimal representation of the encoded tag head is:
0xd9_d9_f9.

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 0xf9.. 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 [STD94] has already determined that no valid CBOR encodings result in valid UTF-16.

Data Item:
tagged byte string

Semantics:
indicates that the file starts with a CBOR-Labeled Non-CBOR Data label.

4.3. CBOR Tags for CoAP Content-Format Numbers

IANA is requested to allocate the tag numbers 1668546817 (0x63740101) to 1668612095 (0x6374ffff) as follows:

Data Item:
byte string or any CBOR data item (see Appendix B of RFC XXXX)

Semantics:
the representation of content-format $ct < 65025$ is indicated by tag number
 $TN(ct) = 0x63740101 + (ct / 255) * 256 + ct \% 255$

Reference:
RFC XXXX

The Registry for Content-Formats of [IANA.core-parameters] has been defined in Section 12.3 of [RFC7252].

5. References

5.1. Normative References

- [C] International Organization for Standardization, "Information technology Programming languages C", ISO/IEC 9899:2018, Fourth Edition, June 2018, <<https://www.iso.org/standard/74528.html>>.
- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

5.2. Informative References

- [file] Wikipedia, "file (command)", 20 January 2021, <https://en.wikipedia.org/wiki/File_%28command%29>.
- [I-D.ietf-cose-cbor-encoded-cert] Mattsson, J. P., Selander, G., Raza, S., Höglund, J., and M. Furuheid, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.txt>>.
- [I-D.ietf-rats-eat] Lundblade, L., Mandyam, G., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-12, 24 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-eat-12.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-21, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-21.txt>>.
- [IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <<https://www.iana.org/assignments/cbor-tags>>.

- [IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters",
<<https://www.iana.org/assignments/core-parameters>>.
- [MAGIC] Ritchie, D., "archive (library) file format", in Bell Labs, Unix Programmer's Manual, First Edition: File Formats, 3 November 1971,
<<https://www.bell-labs.com/usr/dmr/www/man51.pdf#page=4>>.
- [RFC20] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969,
<<https://www.rfc-editor.org/info/rfc20>>.
- [RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, DOI 10.17487/RFC2781, February 2000,
<<https://www.rfc-editor.org/info/rfc2781>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013,
<<https://www.rfc-editor.org/info/rfc6838>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014,
<<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016,
<<https://www.rfc-editor.org/info/rfc8017>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017,
<<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018,
<<https://www.rfc-editor.org/info/rfc8428>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

- [RFC9177] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", RFC 9177, DOI 10.17487/RFC9177, March 2022, <<https://www.rfc-editor.org/info/rfc9177>>.
- [X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, February 2021.

Appendix A. Advice to Protocol Designer

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 ([I-D.ietf-cose-cbor-encoded-cert]) or CBOR format, a recorded sensor reading stored for later transmission, or a COVID-19 vaccination certificate that needs to be displayed in QR code form.

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

The Labeled CBOR Sequence can be slightly 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 Labeled CBOR 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. The three byte entry is not of the format the program would normally have processed, so it may be a surprise. On the other hand, CBOR parsers are generally tolerant of tags that appear: many of them will process extra tags, making unknown tags available as meta information. A program that is not expecting those tags may just ignore those extra tags.

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 [RFC8017] 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 ([X.690], 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 stored 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 files or similar units of storage, 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 B for media-types with CoAP Content-Format numbers.

Here are some considerations in the form of survey questions:

A.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 stored 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 Labeled CBOR Sequence format should be adopted for the stored format.

A.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 Labeled CBOR Sequence format may be easily retrofitted.

The program involved may throw errors or warnings on the Labeled CBOR Sequence if they have not yet been updated, but this may not be a problem.

There are situations where multiple objects may be concatenated into a single file. If each object is preceded by a Labeled CBOR Sequence label then there may be multiple such labels in the file.

A protocol based on CBOR Sequences may specify that Labeled CBOR Sequence labels can occur within a CBOR Sequence, possibly even to switch to data items following in the sequence that are of a different type.

If the CBOR Sequence based protocol does not define the semantics for or at least tolerate embedded labels, care must be taken when concatenating Labeled CBOR Sequences to remove the label from all but the first part.

As an example from legacy PEM encoded PKIX certificates, many programs accept a series of PKIX certificates in a single file in order to set up a certificate chain. The file would contain not just the End-Entity (EE) certificate, but also any subordinate certification authorities (CA) needed to validate the EE. This mechanism actually only works for PEM encoded certificates, and not DER encoded certificates. One of the reasons for this specification is to make sure that CBOR encoded certificates do not suffer from this problem.

As an example of mixing of types, some TLS server programs also can accept both their PEM encoded private key, and their PEM encoded certificate in the same file.

If only one item is ever expected in the file, the use of Labeled CBOR Sequence may present an implementation hurdle to programs that previously just read a single data item and used it.

A.3. Are there tags at the start?

If the Protocol expects to use other tags at its top-level, then the use of the CBOR Tag Wrapped format may be easy to explain at the same place in the protocol description.

Appendix B. CBOR Tags for CoAP Content Formats

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.

Outside of a transfer protocol that indicates the Content-Format for a representation, it may be necessary to identify the Content-Format of the representation when it is stored in a file, in firmware, or when debugging.

This specification allocates CBOR tag numbers 1668546817 (0x63740101) to 1668612095 (0x6374FFFF) for the tagging of representations of specific content formats.

Using tags from this range, a byte string that is to be interpreted as a representation of Content-Format number *ct*, with $ct < 65025$ (255×255), can be identified by enclosing it in a tag with tag number $TN(ct)$ where:

$$TN(ct) = 0x63740101 + (ct / 255) * 256 + ct \% 255.$$

(where +, *, / and % stand for integer addition, multiplication, division and remainder as in the programming language C [C].)

This formula avoids the use of zero bytes in the representation of the tag number.

Note that no tag numbers are assigned for Content-Format numbers in the range 65025 ct 65535. (This range is in the range reserved by Section 12.3 of [RFC7252] for experimental use. The overlap of 25 code points between this experimental range with the range this appendix defines tag numbers for can be used for experiments that want to employ a tag number.)

Exceptionally, when used immediately as tag content of one of the tags 55799, 55800, or 55801, the tag content is as follows:

Tag 55799 (Section 2.2): One of:

1. The CBOR data item within the representation (without byte string wrapping). This only works for Content Formats that are represented by a single CBOR data item in identity content-coding.
2. The data items in the CBOR sequence within the representation, without byte string wrapping, but wrapped in a CBOR array. This works for Content Formats that are represented by a CBOR sequence in identity content-coding.

Tags 55800 (Section 2.3) or 55801 (Appendix D): the byte string 'BOR', signifying that the representation of the given content-format follows in the file, in the way defined for these tags.

B.1. Content-Format Tag Examples

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

- * As discussed in Section 2.2.1, Content-Format 112 stands for media type application/senml+cbor (no parameters). The corresponding tag number is TN(112) = 1668546929.

So the following CDDL snippet can be used to identify application/senml+cbor representations:

```
senml-cbor = #6.1668546929(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 272 stands for media type application/missing-blocks+cbor-seq, a CBOR sequence [RFC9177].

The corresponding tag number is TN(272) = 1668547090.

So the following CDDL snippet can be used to identify application/missing-blocks+cbor-seq representations as embedded in a CBOR byte string:

```
missing-blocks = #6.1668547090(bstr)
```

Appendix C. 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 D. Using CBOR Labels for non-CBOR data

The CBOR-Labeled non-CBOR data method is appropriate for adding a magic number to a non-CBOR data format, particularly one that can be described by a Content-Format tag (Appendix B).

This method prepends a CBOR data item to the non-CBOR data; this data item is called the "header" and, similarly to the Labeled CBOR-Sequence label, consists of two nested tags around a constant byte string for a total of 12 bytes.

1. The outer tag is the CBOR-Labeled Non-CBOR Data tag, 55801.
2. The inner tag is a CBOR tag, from the First Come First Served space, that uniquely identifies the CBOR Protocol. As with CBOR Tag Wrapped, the use of a four-byte tag is encouraged that encodes without zero bytes.
3. The tag content is a three byte CBOR byte string containing 0x42_4F_52 ('BOR' in diagnostic notation).

The outer tag in the label identifies the file as being file as being prefixed by a non-CBOR data label, and does so with all the desirable properties explained in Section 3.4.6 of [STD94]. Specifically, it does not appear to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The inner tag in the label identifies which non-CBOR Protocol is used.

The inner tag content is a constant byte string which is represented as 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three-byte string 0x42_4f_52 ('BOR' in diagnostic notation).

The actual non-CBOR Protocol data then follow directly appended to the CBOR representation of the header. This allows the application to trivially remove the header item with the two nested tags and the byte string.

As with the Labeled CBOR Sequence {#sequences}, this choice of the tag content places the ASCII characters "CBOR" prominently into the header.

D.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 TN(432) = 1668547250.

So the following CDDL snippet can be used to identify a CBOR-Labeled non-CBOR data for application/td+json representations:

```
td-json-header = #6.55801(#6.1668547250('BOR'))
```

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

The corresponding tag number is TN(11050) = 1668557910.

So the following CDDL snippet can be used to identify a CBOR-Labeled non-CBOR data for application/json representations compressed in deflate content-coding:

```
json-deflate-header = #6.55801(#6.1668557910('BOR'))
```

Appendix E. 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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 April 2022

M. Richardson
Sandelman Software Works
C. Bormann
Universität Bremen TZI
22 October 2021

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.

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
2. Terminology	3
3. Protocol	3
3.1. Three Forms	3
3.1.1. Addresses	3
3.1.2. Prefixes	3
3.1.3. Interface Definition	4
3.2. IPv6	4
3.3. IPv4	5
4. Tag validity	6
4.1. Deterministic Encoding	6
4.2. Encoder Considerations for Prefixes	6
4.3. Decoder Considerations for Prefixes	7
4.3.1. Example implementation	7
5. CDDL	8
6. Security Considerations	9
7. IANA Considerations	10
7.1. Tag 54 - IPv6	10
7.2. Tag 52 - IPv4	10
7.3. Tags 260 and 261	10
8. References	10
8.1. Normative References	10
8.2. Informative References	11
Appendix A. Changelog	11
Acknowledgements	11
Authors' Addresses	11

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'20010db81234deedbeefcafeacefeed')

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'20010db81234deedbeefcafeacefeed', 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'fe8000000000020202fffffe030303', 64, 'eth0'])
```

with a numeric zone identifier:

```
54([h'fe8000000000020202fffffe030303', 64, 42])
```

An IPv6 link-local address without a prefix length:

```
54([h'fe8000000000020202fffffe030303', 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')
```

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:

```

ip-address-or-prefix = ipv6-address-or-prefix /
                        ipv4-address-or-prefix

ipv6-address-or-prefix = #6.54(ipv6-address /
                                ipv6-address-with-prefix /
                                ipv6-prefix)
ipv4-address-or-prefix = #6.52(ipv4-address /
                                ipv4-address-with-prefix /
                                ipv4-prefix)

ipv6-address = bytes .size 16
ipv4-address = bytes .size 4

ipv6-address-with-prefix = [ipv6-address,
                            ipv6-prefix-length / null,
                            ?ip-zone-identifier]
ipv4-address-with-prefix = [ipv4-address,
                            ipv4-prefix-length / null,
                            ?ip-zone-identifier]

ipv6-prefix-length = 0..128
ipv4-prefix-length = 0..32

ipv6-prefix = [ipv6-prefix-length, ipv6-prefix-bytes]
ipv4-prefix = [ipv4-prefix-length, ipv4-prefix-bytes]

ipv6-prefix-bytes = bytes .size (uint .le 16)
ipv4-prefix-bytes = bytes .size (uint .le 4)

ip-zone-identifier = uint / text

```

Figure 1: CDDL types for tags 54 and 52

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.

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

8.2. Informative References

- [IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <<https://www.iana.org/assignments/cbor-tags>>.
- [RFC4001] Daniele, M., Haberman, B., Routhier, S., and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 4001, DOI 10.17487/RFC4001, February 2005, <<https://www.rfc-editor.org/info/rfc4001>>.
- [RFC4007] Deering, S., Haberman, B., Jinmei, T., Nordmark, E., and B. Zill, "IPv6 Scoped Address Architecture", RFC 4007, DOI 10.17487/RFC4007, March 2005, <<https://www.rfc-editor.org/info/rfc4007>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7042] Eastlake 3rd, D. and J. Abley, "IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters", BCP 141, RFC 7042, DOI 10.17487/RFC7042, October 2013, <<https://www.rfc-editor.org/info/rfc7042>>.

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

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Carsten Bormann
Universität Bremen TZI
Germany

Email: cabo@tzi.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

C. Bormann
Universität Bremen TZI
20 April 2022

Packed CBOR
draft-ietf-cbor-packed-05

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949 == STD 94) 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 uncompress the compressed form to make use of the data.

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

Note to Readers

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

Status of This Memo

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 October 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

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

1. Introduction

The Concise Binary Object Representation (CBOR, [STD94]) 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 [RFC1951] can work well for CBOR encoded data items, their disadvantage is that the receiver needs to uncompress the compressed form to make use of the 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.

This document defines the Packed CBOR format by specifying the transformation from a Packed CBOR data item to the original CBOR data item; it does not define an algorithm for a packer. Different packers can differ in the amount of effort they invest in arriving at a minimal packed form; often, they simply employ the sharing that is natural for a specific application.

Packed CBOR can make use of 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.

Current set: The packing tables in effect at the data item under consideration.

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

The definitions of [STD94] apply. Specifically: The term "byte" is used in its now customary sense as a synonym for "octet"; "byte strings" are CBOR data items carrying a sequence of zero or more (binary) bytes, while "text strings" are CBOR data items carrying a sequence of zero or more Unicode code points, encoded in UTF-8 [STD63].

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). Such an index may be derived 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.

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

Table 1: Referencing Shared Values

As examples in CBOR diagnostic notation (Section 8 of [STD94]), 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 -- "zigzag encoding" -- 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 CBOR integers can grow to very large (or very 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 tag content: An integer turns the tag into a shared item reference, whereas a string or container (map or array) turns it into a prefix reference (see Table 2). Note also that the tag content of Tag 6 may itself be packed, so it may need to be unpacked to make this determination.

2.3. Referencing Affix Items

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

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

Table 2: Referencing Prefix Values

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

Table 3: Referencing Suffix Values

Affix data items are referenced by using the data items in Table 2 and Table 3. The tag number indicates the table used (prefix or suffix) and a table index (an unsigned integer); the tag content 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 to the rump array, while the elements from a suffix are appended. 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.

NOTE: One application of the rule for prefix references is to supply default values out of a dictionary, which can then be overridden by the entries in the map supplied as the rump value. Note that this pattern provides no way to remove a map entry from the prefix table entry.

- * 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 packing.

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 two thirds of the one-byte simple values. Since the objective is compression, this is warranted only based on a 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.

NOTE: The present specification does nothing to help with the packing of CBOR sequences [RFC8742]; maybe such a specification should be added.

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.

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

	array, map		
216-223	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed
27647-28671	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed
1811940352-1879048191	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed

Table 4: Values for Tag Numbers

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

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

Table 5: Simple Values

5. Security Considerations

The security considerations of [STD94] apply.

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

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

6. References

6.1. Normative References

- [IANA.cbor-simple-values]
IANA, "Concise Binary Object Representation (CBOR) Simple Values",
<<https://www.iana.org/assignments/cbor-simple-values>>.
- [IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags",
<<https://www.iana.org/assignments/cbor-tags>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

6.2. Informative References

- [RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <<https://www.rfc-editor.org/info/rfc1951>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

- [RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <<https://www.rfc-editor.org/info/rfc8742>>.
- [STD63] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/info/rfc3629>>.

Appendix A. Examples

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

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

Figure 2: Example original CBOR data item

```

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

```

Figure 3: Example packed CBOR data item

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

```

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

```

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

```
    "writable": true,
    "@type": [
      "Property"
    ]
  },
  {
    "links": [
      {
        "href":
          "http://192.168.1.103:8445/wot/thing/MyLED/ledOnOff",
        "mediaType": "application/json"
      }
    ],
    "outputData": {
      "valueType": {
        "type": "boolean"
      }
    },
    "name": "ledOnOff",
    "writable": true,
    "@type": [
      "Property"
    ]
  },
  {
    "links": [
      {
        "href":
          "http://192.168.1.103:8445/wot/thing/MyLED/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": [
229({simple(2): [{simple(3): 227("Red"), simple(4): simple(5)}],
simple(0): 228("Red")}),
229({simple(2): [{simple(3): 227("Green"), simple(4): simple(5)}],
simple(0): 228("Green")}),
229({simple(2): [{simple(3): 227("Blue"), simple(4): simple(5)}],
simple(0): 228("Blue")}),
229({simple(2): [{simple(3): 227("White"), simple(4): simple(5)}],
simple(0): "rgbValueWhite"}),
{simple(2): [{simple(3): 226("ledOnOff"), simple(4): simple(5)}],
simple(6): {simple(10): {simple(11): "boolean"}}, simple(0):
"ledOnOff", simple(9): true, simple(1): simple(8)},
{simple(2): [{simple(3): 226("colorTemperatureChanged"),
simple(4): simple(5)}], simple(6): simple(7), simple(0):
"colorTemperatureChanged", simple(1): ["Event"]},
simple(1): "Lamp", "id": "0", "base": 225(""),
"@context": 6("2:8444/wot/w3c-wot-td-context.jsonld")})])

```

Figure 5: Example packed CBOR data item

Acknowledgements

CBOR packing was originally invented with the rest of CBOR, but did not make it into [RFC7049], the predecessor of [STD94]. Various attempts to come up with a specification over the years didn't proceed. In 2017, Sebastian Käbisch proposed investigating compact representations of W3C Thing Descriptions, which prompted the author to come up with essentially the present design.

Author's Address

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 20 November 2021

C. Bormann
Universität Bremen TZI
B. Gamari
Well-Typed
H. Birkholz
Fraunhofer SIT
19 May 2021

Concise Binary Object Representation (CBOR) Tags for Time, Duration, and
Period
draft-ietf-cbor-time-tag-00

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 individual submission that led to 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 were believed will stay unchanged during further development of the draft. This version was provided to aid the registration of the CBOR tag immediately needed. Further versions of the individual submission made use of the IANA allocations registered and made other editorial updates. Now a WG document, future versions could re-introduce some of the material from the initial submission, 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 20 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. 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

9.1. Normative References	10
9.2. Informative References	11
Acknowledgements	11
Authors' Addresses	11

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

2. Objectives

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

- * Additional resolution for epoch-based time (as in tag 1). CBOR tag 1 only provides for integer and up to binary64 floating point representation of times, limiting resolution to approximately microseconds at the time of writing (and progressively becoming worse over time).
- * Indication of time scale. Tags 0 and 1 are for UTC; however, some interchanges are better performed on TAI. Other time scales may be registered once they become relevant (e.g., one of the proposed successors to UTC that might no longer use leap seconds, or a scale based on smeared leap seconds).

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

- * Direct representation of natural platform time formats. Some platforms use epoch-based time formats that require some computation to convert them into the representations allowed by tag 1; these computations can also lose precision and cause ambiguities. (TBD: The present specification does not take a position on whether tag 1 can be "fixed" to include, e.g., Decimal or BigFloat representations. It does define how to use these with the extended time format.)
- * Additional indication of intents about the interpretation of the time given, in particular for future times. Intents might include information about time zones, daylight savings times, etc.

Additional tags are defined for durations and periods.

3. Time Format

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

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

- * a higher precision time offset to be added to the base time,
- * a reference time scale and epoch different from the default UTC and 1970-01-01
- * information about clock quality parameters, such as source, accuracy, and uncertainty

Future keys may add:

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

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

3.1. Key 1

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

3.2. Keys 4 and 5

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

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

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

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

Table 1: Key for decimally scaled Fractions

3.4. Key -1: Time Scale

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

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

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

```
t      = t      - 2208988800
utc    ntp

t      = t      + 315964819
tai    gps
```

Figure 1: Converting Common Offset Time Scales

3.5. Clock Quality

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

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

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

3.5.1. ClockClass (Key -2)

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

```
enum approx48 + |_2cdotlog {accovers} - epsilon_|
  acc              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:

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


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

6. CDDL typenames

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

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

Figure 3: Recommended type names for CDDL

7. IANA Considerations

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

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

Table 2: Values for Tags

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

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

8. Security Considerations

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

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

9. References

9.1. Normative References

- [GUM] Joint Committee for Guides in Metrology, "Evaluation of measurement data Guide to the expression of uncertainty in measurement", JCGM 100:2008, September 2008, <<https://www.bipm.org/en/publications/guides/gum.html>>.
- [IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <<http://www.iana.org/assignments/cbor-tags>>.
- [IEEE1588-2008] IEEE, "1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", July 2008, <<http://standards.ieee.org/findstds/standard/1588-2008.html>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [SI-SECOND] International Organization for Standardization (ISO), "Quantities and units Part 3: Space and time", ISO 80000-3, 1 March 2006.

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

9.2. Informative References

- [RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <<https://www.rfc-editor.org/info/rfc3161>>.
- [RFC8575] Jiang, Y., Ed., Liu, X., Xu, J., and R. Cummings, Ed., "YANG Data Model for the Precision Time Protocol (PTP)", RFC 8575, DOI 10.17487/RFC8575, May 2019, <<https://www.rfc-editor.org/info/rfc8575>>.

Acknowledgements

Authors' Addresses

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

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

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

Email: ben@well-typed.com

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

Email: henk.birkholz@sit.fraunhofer.de