Concise Binary Object Representation Maintenance and Extensions J. Caballero
Internet-Draft R. Berjon
Intended status: Informational IPFS Foundation
Expires: 23 November 2025 22 May 2025

The tag-42 profile of CBOR Core
draft-caballero-cbor-cborc42-00

Abstract

This document defines a strict profile of CBOR Core (CBOR/c) intended for use with the special tag 42. Like the CBOR Core it profiles, "CBOR/c-42" can also be used as an internet-scale serialization for JSON, and is optimized for objects that compose into a directed acyclical graph. Since CBOR/c-42 objects link to one another by hash-based identifiers, deterministic encoding is mandated to verify dereferenced links and encode new ones.

This document mainly targets CBOR tool developers and those downstream users who would like to precisely configure their tools. While full support in CBOR tools would be ideal and is already possible in some highly configurable parsing libraries, ALDRs can help close the delta by sidestepping the biggest interoperability stumbling blocks; see Appendix C for details.

About This Document

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

The latest revision of this draft can be found at https://ipfs-tech.github.io/cborc42/draft-caballero-cbor-cborc42.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-caballero-cbor-cborc42/.

Discussion of this document takes place on the Concise Binary Object Representation Maintenance and Extensions mailing list (mailto:cbor@ietf.org), which is archived at https://www.ietf.org/mail-archive/web/cbor/current/maillist.html. Subscribe at https://www.ietf.org/mailman/listinfo/cbor/.

Source for this draft and an issue tracker can be found at https://github.com/ipfs-tech/cborc42.

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

The developer ecosystem around the Interplanetary File System, a distributed system file and document handling, has long made structural usage of its own home-grown CBOR profile, dating from the early days of the CBOR working group and fine-tuned over the years in community/internal venues. Configuring CBOR tooling in various languages to decode this data and encode new data conformantly has been a challenge, and a unified specification (updated to modern terminology, as the CBOR working group has iterated and evolved so much in the intervening years) is set out in this document.

Note: unlike the CBOR/c specification, no opinion on best practices for hashing or signing mechanisms is expressed here, and will be addressed in separate documents, if at all.

1.1. Design Goals

The primary goal of this specification is enabling application developers to configure CBOR tooling for this profile, and for CBOR tooling to support such configuration, in as language-agnostic a way as possible. The historical design of this profile was to maximize determinism and simplicity for an internet-scale directed acyclical graph of CBOR documents linked to one another by binary hashes. These simple content identifiers, defined in Appendix A, are always expressed as bytestrings of tag 42 (similar in design to [RFC6920] Named Information Hashes). All other tags, and many major and minor types, are forbidden to reduce ambiguity, and developers are encouraged to express many kinds of data at higher layers by using the supported types (such as strings or bytestrings).

1.2. Requirements Language

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.

1.3. Common Definitions

* This document uses the conventions defined in CDDL [RFC8610] for expressing the type of CBOR [RFC8949] data items.

* Examples showing CBOR data, are expressed in "diagnostic notation" as defined in Section 8 of [RFC8949].

* The term "CBOR object" is equivalent to "CBOR data item" used in [RFC8949].

* The term "CBOR Core" is in this document abbreviated to "CBOR/c".

2. Specification

This section describes how CBOR/c-42 subsets CBOR and differs from a standard CDE encoding.

2.1. Supported CBOR Objects

CBOR	Comment
int	Integer
bigint	Big integer
float	64-bit [IEEE754] numbers ONLY
tstr	Text string encoded as UTF-8 [RFC3629]
bstr	Byte string
[]	Array
{}	Map
tag 42	See Appendix A; no other tags allowed
bool	Boolean true and false (major type 7)
null	Represents a null object (major type 7)

Table 1

2.2. Deterministic Encoding Scheme Profile

As in CBOR/c, deterministic encoding is mandatory. The encoding scheme adheres to Section 4.2 of [RFC8949], but adds a few constraints (denoted below by RFC+), where this RFC offers choices. The following list contains a summary of the deterministic encoding rules:

* RFC+: Floating-point and integer objects MUST be treated as distinct types regardless of their numeric value. This is compliant with Rule 2 in Section 4.2.2 of [RFC8949].

* RFC: Integers, represented by the int and bigint types, MUST use the int type if the value is between -2^64 and 2^64-1, otherwise the bigint type MUST be used.

  - Appendix B.1 features a list of integer sample values and their expected encoding.

* RFC+: UNLIKE CBOR/c and standard CDE encoding, floating-point numbers MUST always be encoded using the longest [IEEE754] variant. Appendix B.2 features a list of floating-point sample values and their expected encoding.

* RFC+: NaN values with payloads (like f97e01), or having the most significant bit set ("signaling"), MUST be rejected. See also Appendix B.4 for invalid NaN variants.

* RCF+: UNLIKE CBOR/c and standard CDE encoding, map keys MUST be typed as strings; no other types are allowed as map keys.

* RFC: Map keys MUST be sorted in the bytewise lexicographic order of their deterministic encoding. Duplicate keys (i.e. keys with identical determinstic bytestring values) MUST be rejected; note that semantic equivalence is not tested. As per the "Canonical CBOR" section (3.9) in [RFC7049], the following represents a properly sorted map: { "a": ... , "b": ... , "aa": ... }

* RFC+: Since CBOR encodings according to this specification maintain uniqueness, there are no specific restrictions or tests needed in order to determine map key equivalence. As an (extreme) example, the floating-point numbers 0.0 and -0.0, and the integer number 0 could all get force-typed as three distinct strings (0.0, -0.0, and 0) without colliding.

* RFC: Indefinite length objects MUST be rejected.

2.3. CBOR Tool Requirements

As is the case with CBOR/c, CBOR/c-42 is also designed to allow hashing and signing in raw form. Because of the reduced range of types and tags, many of the tooling requirements made of CBOR/c are not necessary with CBOR/c-42; what follows is an edited version of the requirements proposed by Rundgren’s current draft.

To make "raw" signing safe and verification of such signatures practical, CBOR tooling capable of the following is required:

* It MUST be possible to find out the type of a CBOR object, before it is accessed.

* It MUST be possible to add, delete, and update the contents of CBOR map and array objects, of decoded CBOR data.

* It MUST be possible to reserialize decoded CBOR data, be it updated or not.

* Irrespective of whether CBOR data is decoded, updated, or created programmatically, deterministic encoding MUST be maintained.

* Invalid or unsupported CBOR constructs, as well as CBOR data not adhering to the deterministic encoding scheme MUST be rejected. See also Appendix D and Appendix B.4.

2.3.1. Protocol Primitives

To facilitate cross-platform protocol interoperability, implementers of CBOR/c-42 compatible tools SHOULD include decoder API support for the following primitives.

CBOR	Primitive	Comment	Note
int	Int8	8-bit signed integer	1
uint	Uint8	8-bit unsigned integer	1
int	Int16	16-bit signed integer	1
uint	Uint16	16-bit unsigned integer	1
int	Int32	32-bit signed integer	1
uint	Uint32	32-bit unsigned integer	1
int	Int64	64-bit signed integer	1
uint	Uint64	64-bit unsigned integer	1
integer	BigInt	Integer of arbitrary size	2
float	Float64	64-bit floating-point number	
bool	Boolean	Boolean	
null	Null	Null	4
#7.nnn	Simple	ONLY null, false, true	5
tstr	String	Text string	
bstr	Bytes	Byte string	
See note	EpochTime	Time object expressed as a number	6
See note	DateTime	Time object expressed as a text string	6

Table 2

1. Range testing MUST be performed using the traditional ranges for unsigned respectively two-complement numbers. That is, a hypothetical getUint8() MUST reject numbers outside of 0 to 255, whereas a hypothetical getInt8(), MUST reject numbers outside of -128 to 127.

2. Note that a hypothetical getBigInt() MUST also accept CBOR int objects since int is used for integers that fit in CBOR major type 0 and 1 objects. See also Appendix B.1 and Appendix D.

3. Some platforms do not natively support float32 and/or float16. In this case a hypothetical getFloat16() would need to use a bigger floating-point type for the return value. Note that a hypothetical getFloat16() MUST reject encountered Float32 and Float64 objects. See also Appendix C.

4. Since a CBOR null typically represents the absence of a value, a decoder MUST provide a test-function, like isNull().

5. Simple values in CBOR and CBOR/c include the ranges 0-23 and 32-255, all but three of which are invalid in CBOR/c-42; however, the capability to refer to boolean values (i.e. true and false) and null as major-type 7 simple values MUST be supported to guarantee interoperability with CBOR tooling generally.

6. Since CBOR lacks a native-level time object, Section 3.4 of [RFC8949] introduces two variants of time objects using the CBOR tags 0 and 1, neither of which are supported by the CBOR/c-42 data model for historical interoperability reasons. To support time encoding stably, it is RECOMMENDED that EpochTime and/or DateTime types in input be force-typed as strings at the application level or at the ALDR level. Interoperability with other tooling may be difficult to achieve if support for these APIs is desired, and validating dates at higher layers may introduce new security issues at higher layers.

(Note that the preceding is a strict subset of the protocol primitives enumerated by CBOR/c.)

If a call does not match the underlying CBOR type, the call MUST be rejected.

Due to considerable variations between platforms, corresponding encoder API support does not appear to be meaningful to specify in detail: Java doesn’t have built-in support for unsigned integers, whereas JavaScript requires the use of the JavaScript BigInt type for dealing with 64-bit integers, etc.

2.3.2. Media Type

Protocols transmitting CBOR/c-42 over HTTP interfaces are RECOMMENDED to send all CBOR/c-42 data with a media type header of application/cbor.

2.3.3. Diagnostic Notation

Compliant CBOR/c implementations SHOULD include support for bi-directional diagnostic notation, to facilitate:

* Generation of developer-friendly debugging and logging data

* Easy creation of test and configuration data

Note that decoders for diagnostic notation, MUST always produce deterministically encoded CBOR data, compliant with this specification. This includes automatic sorting of map keys as well.

The supported notation is compliant with a subset of Section 8 of [RFC8949] (b32' and encoding indicators were left out), but adds a few items to make diagnostic notation slightly more adapted for parsing, like single-line comments:

CBOR	Syntax	Comment	Notes
/ comment text /	Multi-line comment. Multi-line comments are treated as whitespace and may thus also be used between CBOR objects.	6	
# comment text	Single-line comment. Single-line comments are terminated by a newline character ('\n') or EOF. Single-line comments may also terminate lines holding regular CBOR items.	6	
integer	{sign}{0b|0o|0x}n	Arbitrary sized integers without fractional components or exponents. See also CBOR integer encoding. For input data in diagnostic notation, binary, octal, and hexadecimal notation is	1, 2

		also supported by prepending numbers with 0b, 0o, and 0x respectively. The latter also permit arbitrary insertions of '_' characters between digits to enable grouping of data like 0b100_000000001.	
float	{sign}n.n{e±n}	Floating point values MUST include a decimal point and at least one fractional digit, whereas exponents are optional.	1, 2
float	NaN	Not a number.	
float	{sign}Infinity	Infinity.	2
bstr	h'hex-data'	Byte data provided in hexadecimal notation. Each byte MUST be represented by two hexadecimal digits.	3
bstr	b64'base64-data'	Byte data provided in base64 or base64URL notation. Padding with '=' characters is optional.	3, 6
bstr	'text'	Byte data provided as UTF-8 encoded text.	4, 5, 6
bstr	<< object... >>	Construct holding zero or more comma-separated CBOR objects that are subsequently wrapped in a byte string.	6
tstr	"text"	UTF-8 encoded text string.	4, 5
bool	true|false	Boolean value.	

null	null	Null value.	
[]	[object...]	Array with zero or more comma-separated CBOR objects.	
{}	{ key:value... }	Map with zero or more comma-separated key/value pairs. Key and value pairs are expressed as CBOR objects, separated by a ':' character.	
#6.nnn	n(object)	Tag holding a CBOR object.	1
#7.nnn	simple(n)	Simple value.	1
,	Separator character for CBOR sequences.	6	

Table 3

The letter n in the Syntax column denotes one or more digits. The optional {sign} MUST be a single hyphen ('-') character. Input only: between the quotes, the whitespace characters (' ', '\t', '\r', '\n') are ignored. Input only: the control characters '\t' and '\n' inside of string quotes become a part of the text. For normalizing line terminators, a single '\r' or the combination '\r\n' MUST (internally) be rewritten as '\n'. To avoid getting newline characters ('\n') included in multi-line text strings, a line continuation marker consisting of a backslash ('') immediately preceding the newline may be used. Text strings may also include JavaScript compatible escape sequences (''', '"', '\', '\b', '\f', '\n', '\r', '\t', '\uhhhh'). Input only. The [PLAYGROUND] is an excellent way of getting acquainted with CBOR and diagnostic notation.

2.3.4. CBOR Sequences

Decoders compliant with this specification MUST support CBOR sequences [RFC8742].

For decoders of "true" (binary) CBOR, there are additional requirements:

It MUST be possible to decode one CBOR object at a time. The decoder MUST NOT make any assumptions about the nature of unread code (it might not even be CBOR).

3. Security Considerations

It is assumed that CBOR/c-42 has no novel security issues compared to CBOR Deterministic Encoding as defined in [RFC8949] but the authors would appreciate any hypotheses or evidence to the contrary.

It should be noted that there has been to date little implementer feedback on the ALDR suggestions outlined in the appendices. As such, these should be considered as an understudied security surface for the application layer to consider.

4. IANA Considerations

This document has no IANA actions.

5. References

5.1. Normative References

[IEEE754] "IEEE Standard for Floating-Point Arithmetic", IEEE, DOI 10.1109/ieeestd.2019.8766229, ISBN ["9781504459242"], July 2019, <https://doi.org/10.1109/ieeestd.2019.8766229>.

[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/rfc/rfc2119>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/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/rfc/rfc8610>.

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

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

5.2. Informative References

[DASL.ING] Berjon, R. and J. Caballero, "DASL Content Identifiers", 2024, <https://dasl.ing/cid.html>.

[ECMASCRIPT] "ECMAScript 2024 Language Specification", n.d., <https://www.ecma-international.org/publications/standards/Ecma-262.htm>.

[MULTIFORMATS] "Multiformats Community Registry", n.d., <https://github.com/multiformats/multicodec/>.

[PLAYGROUND] "Online CBOR testing tool", n.d., <https://cyberphone.github.io/CBOR.js/doc/playground.html>.

[protobuf] "Encoding rules and MIME type for Protocol Buffers", 2012, <https://www.ietf.org/archive/id/draft-rfernando-protocol-buffers-00.txt>.

[RFC3629] 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/rfc/rfc3629>.

[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/rfc/rfc6256>.

[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/rfc/rfc6920>.

Appendix A. Binary Content Identifiers

A simple hash-based "content identifier" is used to link documents in the graph for which CBOR/c-42 was designed, and tag 42 was registered specifically for those link identifiers in the IANA registry, "Concise Binary Object Representation (CBOR) Tags" created by Section 9.2 (https://datatracker.ietf.org/doc/html/rfc8949#section-9.2) of [RFC8949].

Being able to navigate or generate new links in this graph are strictly unrelated concerns and of course optional for a CBOR/c-42 encoder and decoder, so this entire section is non-normative and provided informationally for the purposes of making less opaque the bytestrings marked by tag 42. Some CBOR/c-42 parsers may want to introspect the tag 42 values, if only to know which dereference to other CBOR/c-42 (or vanilla CBOR) documents.

Note: this describes tag-42 values from the perspective of the CBOR documents in which they are embedded; a simpler, "application developer"-oriented overview of content identifiers can be found at [DASL.ING].

The format consists of a few sigils prepended to a hash of the linked document; there are many possible values for these sigils but these are like CBOR tags, extension points rather than required features of the system. All the sigils and the hash are of variable length, encoded as Self-Delimiting Numeric Values ([RFC6256]). The sequence of segments is as follows:

1. Mandatory padding byte 0x00. This is required for interoperability with non-CBOR CID systems and for historical reasons (see Legacy Support section below).

2. Version byte: 0x01 is the only value worth attempting to parse in a general-purpose tool. 0x00 refers to a legacy form explained below, with 0x02 and 0x03 reserved for potential future use.

3. A contextualizing sigil roughly mapping to content types, with the canonical registry governed by a community registry called [MULTIFORMATS]. The only values that all CBOR/c-42 decoders need to recognize are:

    1. 0x71 - CBOR/c-42

    2. 0x51 - Any other form of CBOR

    3. 0x55 - Raw, unstructured binary

4. A hash-function sigil, drawn from the same registry. Likewise, the vast majority of values here are optional extensions; the required hash functions to be aware of are:

    1. 0x12 - SHA-256

5. A hash-length (as SDNV).

6. The hash (all remaining bytes).

A.1. Legacy Support

Any tag 42 value that does NOT begin with 0x00 can be considered malformed, and any attempt to recuperate legacy links or variations from such a value is entirely optional.

The most common form of legacy data from deprecated encodings is the historical v0 form IPFS content identifiers, which were always 34 bytes long and consisted only of segments 4 (0x12), 5 (0x20, i.e. 32 bytes) and 6 above (a 32-byte SHA256 hash). Prepending 0x00 (padding byte), 0x01 (CID version), 0x70 (DAG-profiled protobuf) turns these into valid "v1" content identifiers, although they still dereference to protobuf objects rather than CBOR objects. For guidance on protobuf deserialization, see protobuf.dev or the relevant [protobuf] draft RFC.

Likewise, some specialized applications that can strictly assume segments 1-3 or 1-5 will be invariant systemwide have been observed to use "truncated" content identifiers, prepending the invariant prefixes only in transformations at point of egress for interoperability purposes. This is not best practice but can also serve as some explanation for the padding byte.

Appendix B. Test Vectors

B.1. Integers

Diagnostic Notation	CBOR Encoding	Comment
0	00	Smallest positive implicit int
-1	20	Smallest negative implicit int

23	17	Largest positive implicit int
-24	37	Largest negative implicit int
24	1818	Smallest positive one-byte int
-25	3818	Smallest negative one-byte int
255	18ff	Largest positive one-byte int
-256	38ff	Largest negative one-byte int
256	190100	Smallest positive two-byte int
-257	390100	Smallest negative two-byte int
65535	19ffff	Largest positive two-byte int
-65536	39ffff	Largest negative two-byte int
65536	1a00010000	Smallest positive four-byte int
-65537	3a00010000	Smallest negative four-byte int

4294967295	1affffffff	Largest positive four-byte int
-4294967296	3affffffff	Largest negative four-byte int
4294967296	1b0000000100000000	Smallest positive eight-byte int
-4294967297	3b0000000100000000	Smallest negative eight-byte int
18446744073709551615	1bffffffffffffff	Largest positive eight-byte int
-18446744073709551616	3bffffffffffffff	Largest negative eight-byte int
18446744073709551616	c249010000000000000000	Smallest positive bigint
-18446744073709551617	c349010000000000000000	Smallest negative bigint

Table 4

B.2. Floating Point Numbers

The textual representation of the values is based on the serialization method for the Number data type, defined by [ECMASCRIPT] with one change: to comply with diagnostic notation (section 8 of [RFC8949]), all values are expressed as floating-point numbers. The rationale for using [ECMASCRIPT] serialization is because it is supposed to generate the shortest and most correct representation of [IEEE754] numbers.

Diagnostic Notation	CBOR/c-42 Encoding	CBOR/c Encoding	Comment
0.0	?	f90000	Zero
-0.0	?	f98000	Negative zero
Infinity	invalid	f97c00	Infinity
-Infinity	invalid	f9fc00	Negative infinity
NaN	invalid	f97e00	Not a number
5.960464477539063e-8	fb3e70000000000000	f90001	Smallest positive subnormal float16
0.00006097555160522461	fb3f0ff80000000000	f903ff	Largest positive subnormal float16
0.00006103515625	fb 3f10000000000000	f90400	Smallest positive float16
65504.0	fb 40effc0000000000	f97bff	Largest positive float16
1.401298464324817e-45	fb36a0000000000000	fa00000001	Smallest positive subnormal float32
1.1754942106924411e-38	fb380fffffc0000000	fa007fffff	Largest positive subnormal float32
1.1754943508222875e-38	fb3810000000000000	fa00800000	Smallest positive float32

3.4028234663852886e+38	fb 47efffffe0000000	fa7f7fffff	Largest positive float32
5.0e-324	fb0000000000000001	fb0000000000000001	Smallest positive subnormal float64
2.225073858507201e-308	fb000fffffffffffff	fb000fffffffffffff	Largest positive subnormal float64
2.2250738585072014e-308	fb0010000000000000	fb0010000000000000	Smallest positive float64
1.7976931348623157e+308	fb7fefffffffffffff	fb7fefffffffffffff	Largest positive float64
-0.0000033333333333333333	fbbecbf647612f3696	fbbecbf647612f3696	Randomly selected number
10.559998512268066	fb40251eb820000000	fa4128f5c1	-"-
10.559998512268068	fb40251eb820000001	fb40251eb820000001	Next in succession
295147905179352830000.0	fb 4430000000000000	fa61800000	268 (diagnostic notation truncates precision)
2.0	fb4000000000000000	f94000	Number without a fractional part
-5.960464477539063e-8	fbbe70000000000000	f98001	Smallest negative subnormal float16
-5.960464477539062e-8	fbbe6fffffffffffff	fbbe6fffffffffffff	Adjacent

			smallest negative subnormal float16
-5.960464477539064e-8	fbbe70000000000001	fbbe70000000000001	-"-
-5.960465188081798e-8	fbbe70000020000000	fab3800001	-"-
0.0000609755516052246	fb3f0ff7ffffffffff	fb3f0ff7ffffffffff	Adjacent largest subnormal float16
0.000060975551605224616	fb3f0ff80000000001	fb3f0ff80000000001	-"-
0.000060975555243203416	fb3f0ff8002000000	fa387fc001	-"-
0.0000610351562499999	fb3f0fffffffffffff	fb3f0fffffffffffff	Adjacent smallest float16
0.0000610351562500001	fb3f10000000000001	fb3f10000000000001	-"-
0.000061035163525957 61	fb3f10000020000000	fa38800001	-"-
65503.99999999999	fb40effbffffffffff	fb40effbffffffffff	Adjacent largest float16
65504.00000000001	fb40effc0000000001	fb40effc0000000001	-"-
65504.00390625	fb40effc0020000000	fa477fe001	-"-
1.4012984643248169e-45	fb369fffffffffffff	fb369fffffffffffff	Adjacent smallest subnormal float32
1.4012984643248174e-45	fb36a0000000000001	fb36a0000000000001	-"-
1.175494210692441e-38	fb380fffffbfffffff	fb380fffffbfffffff	Adjacent largest subnormal float32
1.1754942106924412e-38	fb380fffffc0000001	fb380fffffc0000001	-"-

1.1754943508222874e-38	fb380fffffffffff	fb380fffffffffff	Adjacent smallest float32
1.1754943508222878e-38	fb3810000000000001	fb3810000000000001	-"-
3.4028234663852882e+38	fb47effffdffffff	fb47effffdffffff	Adjacent largest float32
3.402823466385289e+38	fb47effffe0000001	fb47effffe0000001	-"-

Table 5

B.3. Miscellaneous Items

Diagnostic Notation	CBOR Encoding	Comment
true	f5	Boolean true (allowed simple value)
null	f6	Null (allowed simple value)
simple(59)	f83b	Disallowed simple value
59	183b	unsigned integer
-59	383a	signed integer
0("2025-03-30T12:24:16Z")	c074323032352d30332d33	
305431323a32343a31365a	Disallowed ISO date/time	
[1, [2, 3], [4, 5]]	8301820203820405	Array combinations
{ "a": 0, "b": 1, "aa": 2}	a361610161620262616103	Map object
h'48656c6c6f2043424f5221'	4b48656c6c6f2043424f5221	Binary string

" science"	6cf09f9a8020736369656e6365	Text string with emoji

Table 6

B.4. Invalid Encodings

CBOR Encoding	Diagnostic Notation	Comment
a2616201616100	{ "b": 1, "a": 0 }	Improper map key ordering
1900ff	255	Number with leading zero bytes
c34a00010000000000000000	-18446744073709551617	Number with leading zero bytes
Fa41280000	10.5	Not in shortest encoding
c243010000	65536	Incorrect value for bigint
fa7fc00000	NaN	Not in shortest encoding
f97e01	NaN	NaN with payload
f97e00	NaN	NaN disallowed even in shortest encoding
5f4101420203ff	(_ h'01', h'0203')	Indefinite length object
fc		Reserved
f818		Invalid simple value
5b0010000000000000		Extremely large bstr length indicator:

| | | |4503599627370496|
+---+---+---+

Table 7

Appendix C. Configuration and ALDRs

Someone familiar with the long history of deterministic or canonical CBOR will note that the above specification mixes and matches properties from that history of profiling. This creates three major issues for CBOR parsers that are not highly configurable:

1. The drastically reduced set of types and tags, as well as the requirement that map keys be typed as strings, usually require enformcement at the application layer, i.e. as "ALDR"s.

2. Configuring a generic library to _encode_ CBOR according to this profile's map-sorting requirement, when that library does not support [RFC7049] Canonical-CBOR sort mode (sometimes called "legacy" or "lengthfirst"), can be a substantial burden, and may require implementing that sorting algorithm at the application layer if the parser allows preserving map order in input.

3. Issues around float reduction are harder to triage at the application layer, although many ALDRs and applications that use this encoding (such as that of the BlueSky social network and the data model of its underlying "Authenticated Data Protocol") completely sidestep the issue by simply disallowing floats at the CBOR level, or transcoding floats to a "virtual type" at the application layer, e.g. by retyping floats as strings.

Appendix D. Decoding Strictness

When decoding CBOR data encoded without observing the rules defined above, it recommended that validity rules around allowed types and tags, integer reduction, float reduction, and map sorting follow the looser norms set out in [RFC8949]. A CBOR/c-42 application or encoder has no obligation to support re-encoding of such non-profile data according to these looser rules, however, and roundtrip-translation is unlikely to be guaranteed as this was a non-goal of the original design.

Acknowledgments

The authors would like to thank for their guidance over a long period on this and related projects:

* Carsten Bormann

* Paul Hoffman
* Dirk Kutscher
* Volker Mische
* Marcin Rataj
* Rod Vagg

Authors' Addresses

Juan Caballero
IPFS Foundation
Email: bumblefudge@learningproof.xyz

Robin Berjon
IPFS Foundation
Email: robin@berjon.com