Decentralization of the Internet
Internet-Draft
Intended status: Informational
Expires: 18 September 2026

J. Caballero
R. Berjon
IPFS Foundation

17 March 2026

The tag-42 profile of CBOR
draft-caballero-cbor-cbor42-02

Abstract

This document defines a bespoke serialization of CBOR intended for use with the special tag 42 in various end-to-end protocols that came out of the IPFS community. Much of its design dates to the first CBOR RFC and predates much of the terminology and the layered approach to determinism elaborated in later years.

CBOR-42 can be used as an internet-scale serialization for JSON, and is optimized for objects that compose into a directed acyclical graph. Since CBOR-42 objects link to one another by hash-based identifiers tagged "42", deterministic encoding is mandated to verify dereferenced links and encode new ones.

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/cbor42/draft-caballero-cbor-cbor42.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-caballero-cbor-cbor42/.

Discussion of this document takes place on the Decentralization of the Internet mailing list (mailto:din@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/din/. Subscribe at https://www.ietf.org/mailman/listinfo/din/.

Source for this draft and an issue tracker can be found at https://github.com/ipfs-tech/cbor42.

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 serialization, 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: 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 serialization, 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, and for an end-to-end protocol by which this graph grows. These simple binary-hash 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 are forbidden to reduce ambiguity, and developers are encouraged to express many kinds of data at higher layers by using the small set of supported types (such as strings or bytestrings); see the Appendix Application-Level Considerations (Appendix C).

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].

2. Specification

This section describes the CBOR-42 serialization and how it differs from a deterministic encoding defined in the draft BCP on [Determinism].

2.1. Supported CBOR Objects

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

Table 1

2.2. CBOR-42 Serialization

CBOR-42 was designed for determinism (a decade before the generalized CDE deterministic serialization or dCBOR was finalized) and the protocols and applications that it was designed for all mandate its strict encoding. The encoding scheme is most similar to the deterministic encoding, but with some major differences. The following list contains a summary of these differences:

* 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 only by the int type or untagged bytestrings or strings, MUST use the int type if the value is between -2^64 and 2^64-1; otherwise, they can be encoded as bytestrings WITHOUT the bignum tag or as strings, and discrimation from other bytestrings or strings is expected to be handled at the application layer.

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

* Unlike the preferred-plus or CDE serializations, 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.

* 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.

* UNLIKE the preferred-plus or CDE serialzations, map keys MUST be typed as strings; no other types are allowed as map keys.

* Map keys MUST be strings and MUST be sorted "length-first", which (because they are strings) can always be achieved by sorting in bytewise lexicographic order (see [RFC8949] section 4.2.3; deterministic encoding uses the other ordering from section 4.2.1). Duplicate keys (i.e. keys with identical deterministic bytestring values) MUST be rejected. Note that semantic equivalence is not tested when detecting duplicate keys.

  - Since map keys must be strings, the following represents a properly sorted map, whether sorted according to the "Canonical CBOR" algorithm: { "a": ... , "b": ... , "aa": ... }

  - 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.

* Indefinite length objects of any kind MUST be rejected.

2.3. CBOR Tool Requirements

CBOR-42 is designed to allow hashing and signing in raw form.

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, CBOR-42 encoding MUST be maintained, including the less-common sorting of string-keyed maps.

* Invalid or unsupported CBOR constructs, as well as valid CBOR data not adhering to the CBOR-42 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-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
float	Float64	64-bit floating-point number	
bool	Boolean	Boolean	
null	Null	Null	2
#7.nnn	Simple	ONLY null, false, true	3
tstr	String	Text string	
bstr	Bytes	Byte string	
See note	EpochTime	Time object expressed as a number	4
See note	DateTime	Time object expressed as a text string	4

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

3. Simple values in CBOR include the ranges 0-23 and 32-255, all but three of which are invalid in CBOR-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.

4. 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.

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

2.3.2. Media Type

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

2.3.3. CBOR Sequences

Concatenating or streaming CBOR objects is strongly discouraged except in contexts where the application/cbor-seq media type can be used to set decoder expectations appropriately. See [RFC8742] for guidance on streaming best practices.

3. Security Considerations

It is assumed that CBOR-42 has no novel security issues compared to the deterministic serialization as defined in [RFC8949] and the draft BCP on [Determinism] 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 requests 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>.

[Determinism]
"CBOR Serialization and Determinism", n.d., <https://datatracker.ietf.org/doc/draft-ietf-cbor-serialization/>.

[ECMASCRIPT]
"ECMAScript速 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/>.

[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-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 orthogonal concerns and of course optional for a CBOR-42 encoder and decoder, so this entire section is provided informationally for the purposes of making less opaque the bytestrings marked by tag 42. Some CBOR-42 parsers may want to introspect the tag 42 values, if only to know which dereference to other CBOR-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 to expect 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 taken from 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 through 3 or 1 through 5 in the list above 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	1bffffffffffffffff	Largest positive eight-byte int
-18446744073709551616	3bffffffffffffffff	Largest negative eight-byte int

Table 3

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-42 Encoding	CBOR Encoding	Comment
0.0	fb0000000000000000	fb0000000000000000	Zero
-0.0	fb8000000000000000	fb8000000000000000	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	fb3f10000000000000	f90400	Smallest positive float16
65504.0	fb40effc0000000000	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	fb47efffffe0000000	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	fb4430000000000000	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	fbbe6fffffffffff	fbbe6fffffffffff	Adjacent smallest negative subnormal float16
-5.960464477539064e-8	fbbe70000000000001	fbbe70000000000001	-"-
-5.960465188081798e-8	fbbe70000020000000	fab3800001	-"-
0.0000609755516052246	fb3f0ff7ffffffffff	fb3f0ff7ffffffffff	Adjacent largest subnormal float16
0.00006097555516052246	fb3f0ff80000000001	fb3f0ff80000000001	-"-
0.00006097555555243203416	fb3f0ff8002000000	fa387fc001	-"-
0.0000610351562499999	fb3f0fffffffffffff	fb3f0fffffffffffff	Adjacent smallest float16
0.0000610351562500001	fb3f10000000000001	fb3f10000000000001	-"-
0.0000610351635259576	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	fb380fffffffffffff	fb380fffffffffffff	Adjacent smallest float32
1.1754943508222878e-38	fb3810000000000001	fb3810000000000001	-"-
3.4028234663852882e+38	fb47effffdffffff	fb47effffdffffff	Adjacent largest float32
3.402823466385289e+38	fb47effffe0000001	fb47effffe0000001	-"-

Table 4

B.3. Miscellaneous Items

Diagnostic Notation	CBOR-42 Encoding	CBOR Encoding	Comment
true	f5	f5	Boolean true (allowed simple value)
null	f6	f6	Null (allowed simple value)
simple(59)	n/a	f83b	Disallowed simple value
59	183b	183b	unsigned integer
-59	383a	383a	signed integer

```
-------+
   |0("2025-03-30T12:24:16Z")|"2025-03-30T12:24:16Z"    |c074323032352d30332d33    |appli
cation-|
   |                         |                          |                          |level
    |
   |                         |                          |                          |taggi
ng    |
   |                         |                          |                          |assum
ed    |
   +-------------------------+--------------------------+--------------------------+-----
-------+
   |305431323a32343a31365a   |n/a                       |n/a                       |Disal
lowed  |
   |                         |                          |                          |ISO d
ate/   |
```

			time
[1, [2, 3], [4, 5]]	8301820203820405	8301820203820405	Array combinations
{ "a": 0, "b": 1, "aa": 2}	a361610161620262616103	a361610161620262616103	Map object
h'48656c6c6f2043424f5221'	4b48656c6c6f2043424f5221	4b48656c6c6f2043424f5221	Binary string
" science"	6cf09f9a8020736369656e6365	6cf09f9a8020736369656e6365	Text string with emoji

Table 5

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	Only 64-bit floats
c243010000	65536	bigints not allowed
c249010000000000000000	18446744073709551616	bigints not allowed
c349010000000000000000	-18446744073709551617	bigints not allowed
fa7fc00000	NaN	NaNs not allowed
f97e01	NaN	NaNs not allowed

f97e00	NaN	NaNs not allowed
5f4101420203ff	(_ h'01', h'0203')	Indefinite length object
fc		Reserved
f818		Invalid simple value
5b0010000000000000		Extremely large bstr length indicator: 4503599627370496

Table 6

Appendix C. Application-level considerations

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 enforcement mechanisms at the application layer (known in the CBOR WG discussions as "ALDR"s). These are outlined below.

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 encoder can be configured to preserve map order in input.

3. Issues around float reduction are harder to triage at the application layer, although many CBOR-42 applications (such as that of the Bluesky social network and the data model of its underlying "Authenticated Transfer Protocol") completely sidestep the issue by simply disallowing floats at the application level to avoid having to handle them at the CBOR level. Some applications seeking interoperability with these float-free applications transcode floats to a "virtual type" at the application layer, e.g. by retyping floats as strings.

C.1. Bignums and Bytestrings

Unlike in serializations that use major type 2 ("BIGNUMS"), integers larger than the integer basic type are not tagged as such when being encoded as bytestrings. To avoid confusing large integers and other uses of the bytes major type, applications generally use some form of application-level metadata or schema system rather than the CBOR tag: in the case of older forms of IPFS like the UnixFS file system, there is an IPLD schema validation step between the application layer and the CBOR encoding; in the case of BlueSky and the AT Protocol, the equivalent schematization happens at the layer of "lexicons".

C.2. Datetimes

Similarly, CBOR-42 does not use the tag for either CBOR datetime format.

Acknowledgments

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

* Carsten Bormann
* David Buchanan
* Cole Capilongo
* Paul Hoffman
* Dirk Kutscher
* Volker Mische
* Wolf McNally
* Bryan Newbold
* Marcin Rataj
* Anders Rundgren
* Rod Vagg

Authors' Addresses

Juan Caballero
IPFS Foundation

Email: bumblefudge@learningproof.xyz

Robin Berjon
IPFS Foundation
Email: robin@berjon.com