

CBOR
Internet-Draft
Intended status: Best Current Practice
Expires: 8 January 2026

C. Bormann
Universität Bremen TZI
7 July 2025

CBOR Common Deterministic Encoding (CDE)
draft-ietf-cbor-cde-12

Abstract

CBOR (STD 94, RFC 8949) defines "Deterministically Encoded CBOR" in its Section 4.2, providing some flexibility for application specific decisions. To facilitate Deterministic Encoding to be offered as a selectable feature of generic encoders, the present document documents the Best Current Practice for CBOR Common Deterministic Encoding (CDE), which can be shared by a large set of applications with potentially diverging detailed requirements. It also defines the term "Basic Serialization", which stops short of the potentially more onerous requirements that make CDE fully deterministic, while employing most of its reductions of the variability needing to be handled by decoders.

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-cde/>.

Discussion of this document takes place on the Concise Binary Object Representation Maintenance and Extensions (CBOR) Working Group mailing list (<mailto:cbor@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/cbor/>. Subscribe at <https://www.ietf.org/mailman/listinfo/cbor/>.

Source for this draft and an issue tracker can be found at
<https://github.com/cbor-wg/draft-ietf-cbor-cde>.

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 8 January 2026.

Copyright Notice

Copyright (c) 2025 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. Structure of This Document	3
1.2. Conventions and Definitions	4
2. Encoding Choices in CBOR	5
3. CBOR Common Deterministic Encoding (CDE)	7
3.1. CDE Constraints from Preferred Serialization	7
3.2. Additional CDE Constraint from Basic Serialization	10
3.3. Additional CDE Constraint from CDE Itself	10
4. CDDL support	10
5. Security Considerations	11
6. IANA Considerations	11
7. References	12
7.1. Normative References	12
7.2. Informative References	13
Appendix A. Information Model, Data Model and Serialization	14
A.1. Data Model, Encoding Variants and Interoperability with Partial Implementations	15
Appendix B. Application-level Deterministic Representation	17
Appendix C. Implementers' Checklists	20
C.1. Preferred Serialization	21
C.1.1. Preferred Serialization Encoders	21
C.1.2. Decoders and Preferred Serialization	23
C.2. Basic Serialization	23
C.2.1. Basic Serialization Encoders	24
C.2.2. Decoders and Basic Serialization	24

C.3. CDE	24
C.3.1. CDE Encoders	24
C.3.2. CDE-checking Decoders	25
Appendix D. Encoding Examples	25
D.1. Integer Value Examples	25
D.2. Floating Point Value Examples	27
D.3. Failing Examples	29
Appendix E. Examples for Preferred Serialization of Integers . .	30
Appendix F. Example Code for Encoding into 16-bit Floating Point	31
List of Figures	32
List of Tables	32
Acknowledgments	33
Contributors	33
Author's Address	33

1. Introduction

CBOR (STD 94, RFC 8949) defines "Deterministically Encoded CBOR" in its Section 4.2, providing some flexibility for application specific decisions. To facilitate Deterministic Encoding to be offered as a selectable feature of generic encoders, the present document documents the Best Current Practice for CBOR Common Deterministic Encoding (CDE), which can be shared by a large set of applications with potentially diverging detailed requirements. It also defines the term "Basic Serialization", which stops short of the potentially more onerous requirements that make CDE fully deterministic, while employing most of its reductions of the variability needing to be handled by decoders.

1.1. Structure of This Document

After introductory material (this introduction and Section 2), Section 3 defines the CBOR Common Deterministic Encoding (CDE). Section 4 defines Concise Data Definition Language (CDDL) support for indicating the use of CDE. This is followed by the conventional sections for Security Considerations (5), IANA Considerations (6), and References (7).

For use as background material, Appendix A introduces terminology for the layering of models used to describe CBOR.

Instead of giving rise to the definition of application-specific, non-interoperable variants of CDE, this document identifies Application-level Deterministic Representation (ALDR) rules as a concept that is separate from CDE itself (Appendix B) and therefore out of scope for this document. ALDR rules are situated at the application-level, i.e., on top of the CDE, and address requirements

on deterministic representation of application data that are specific to an application or a set of applications. ALDR rules are routinely provided as part of a specification for a CBOR-based protocol, or, if needed, can be provided by referencing a shared "ALDR ruleset" that is defined in a separate document.

The informative Appendix C provides brief checklists that implementers can use to check their CDE implementations. Appendix C.1 provides a checklist for implementing Preferred Serialization. Appendix C.2 introduces "Basic Serialization", a slightly more restricted form of Preferred Serialization that may be used by encoders to hit a sweet spot for maximizing interoperability with partial (e.g., constrained) CBOR decoder implementations. Appendix C.3 further adds constraints to Basic Serialization to arrive at CDE.

Appendix D provides a few examples for CBOR data items in CDE encoding, as well as a few failing examples; Appendix E examines preferred serialization of the number 1 in more detail. For reference by implementers, Appendix F shows an implementation that encodes a floating point number as "half precision" binary16.

1.2. Conventions and Definitions

The conventions and definitions of [STD94] apply. Appendix A provides additional discussion of the terms information model, data model, and serialization.

- * The term "CBOR Application" ("application" for short) is not explicitly defined in [STD94]; this document uses it in the same sense as it is used there, specifically for applications that use CBOR as an interchange format and use (often generic) CBOR encoders/decoders to serialize/ingest the CBOR form of their application data to be exchanged.
- * Similarly, "CBOR Protocol" is used as in [STD94] for the protocol that governs the interchange of data in CBOR format for a specific application or set of applications.
- * "Representation" stands for the process, and its result, of building the representation format out of (information-model level) application data.
- * "Serialization" is used for the subset of this process, and its result, that represents ("serializes") data in CBOR generic data model form into encoded data items. "Encoding" is often used as a synonym when the focus is on that.

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 [BCP14] (RFC2119) (RFC8174) when, and only when, they appear in all capitals, as shown here.

2. Encoding Choices in CBOR

In many cases, CBOR provides more than one way to encode a data item, i.e., to serialize it into a sequence of bytes. This flexibility can provide convenience for the generator of the encoded data item, but handling the resulting variation can also put an onus on the decoder. In general, there is no single perfect encoding choice that is optimal for all applications. Choosing the right constraints on these encoding choices is one element of application protocol design. Having predefined sets of such choices is a useful way to reduce variation between applications, enabling generic implementations.

Section 4.1 of RFC 8949 [STD94] provides a recommendation for a Preferred Serialization. This recommendation is useful for most CBOR applications, and it is a good choice for most applications. Its main constraint is to choose the shortest head (Section 3 of RFC 8949 [STD94]) that preserves the value of a data item.

Preferred Serialization allows indefinite length encoding (Section 3.2 of RFC 8949 [STD94]), which does not express the length of a string, an array, or a map in its head. Supporting both definite length and indefinite length encoding is an additional onus on the decoder; many applications therefore choose not to use indefinite length encoding at all. We call Preferred Serialization with this additional constraint Basic Serialization. Basic Serialization is a common choice for applications that need to further reduce the variability that needs to be handled by decoders, potentially maximizing interoperability with partial (e.g., constrained) CBOR decoder implementations.

These constraints still allow some variation. In particular, there is more than one serialization for data items that contain maps: The order of serialization of map entries is ignored in CBOR (as it is in JSON), so maps with more than one entry have all permutations of these entries as valid Basic Serializations. `_Deterministic Serialization_` builds on Basic Serialization by defining a common (namely, lexicographic) order for the entries in a map. For many applications, ensuring this common order is an additional onus on the generator that is not actually needed, so they do not choose Deterministic Serialization. However, if the objective is minimal effort for the consuming application, deterministic map ordering can be useful even outside the main use cases for Deterministic Serialization that are further discussed in Section 2 of [I-D.bormann-cbor-det].

Table 1 summarizes the increasingly restrictive sets of encoding choices that have been given names in this section.

Set of Encoding Choices	Most Important Constraint	Applications
Preferred	shortest "head" variant	most
Basic	+ definite lengths only	many
<code>_Deterministic_ ("CDE")</code>	+ common map order	specific

Table 1: Constraints on the Serialization of CBOR

Note that the objective to have a deterministic serialization for a specific application data item can only be fulfilled if the application itself does not generate multiple different CBOR data items that represent that same (equivalent) application data item. We speak of the need for Application-level Deterministic Representation (ALDR), and we may want to aid achieving this by the application defining rules for ALDR (see also Appendix B). Where Deterministic Representation is not actually needed, application-level representation rules of course can still be useful to amplify the benefits of Preferred or Basic Serialization.

3. CBOR Common Deterministic Encoding (CDE)

This specification documents the `_CBOR Common Deterministic Encoding_` (CDE) Best Current Practice that is based on the `_Core Deterministic Encoding Requirements_` defined for CBOR in Section 4.2.1 of RFC 8949 [STD94].

Note that this specific set of requirements is elective — in principle, other variants of deterministic encoding can be defined (and have been, now being phased out, as detailed in Section 4.2.3 of RFC 8949 [STD94]). In many applications of CBOR today, deterministic encoding is not used at all, as its restriction of choices can create some additional performance cost and code complexity.

[STD94]'s "Core Deterministic Encoding Requirements" are designed to provide well-understood and easy-to-implement rules while maximizing coverage, i.e., the subset of CBOR data items that are fully specified by these rules, and also placing minimal burden on implementations.

As discussed in Section 2, CDE combines the constraints of Preferred Serialization with a constraint added by Basic Serialization and another constraint added by CDE itself. While many CBOR implementations do set out to provide Preferred Serialization, there is less of a practical requirement to fully conform, as generic CBOR decoders do not normally check for Preferred Serialization. However, an application that relies on deterministic representation, during ingestion of an encoded CBOR data item will often need to employ a "CDE-checking decoder", i.e., a CBOR decoder configured to also check that all CDE constraints are satisfied (see also Appendix C). Here, small deviations from CDE, including deviations from Preferred Serialization, turn into interoperability problems; hence the additional attention of the present document on these constraints.

3.1. CDE Constraints from Preferred Serialization

Section 4.2.2 of RFC 8949 [STD94] picks up on the interaction of extensibility (CBOR tags) and deterministic encoding. CBOR itself uses some tags to increase the range of its basic generic data types, e.g., tags 2/3 extend the range of basic major types 0/1 in a seamless way. Section 4.2.2 of RFC 8949 [STD94] recommends handling this transition the same way as with the transition between different integer representation lengths in the basic generic data model, i.e., by mandating the Preferred Serialization for all integers (Section 3.4.3 of RFC 8949 [STD94]; see also Appendix D.1 and Appendix E).

1. By adopting the encoding constraints from Preferred Serialization, CDE turns this recommendation into a mandate: Integers that can be represented by basic major type 0 and 1 are encoded using the deterministic encoding defined for them, and integers outside this range are encoded using the Preferred Serialization (Section 3.4.3 of RFC 8949 [STD94]) of tag 2 and 3 (i.e., no leading zero bytes).

Most tags capture more specific application semantics and therefore may be harder to define a deterministic encoding for. While the deterministic encoding of their tag internals is often covered by the Core Deterministic Encoding Requirements, the mapping of diverging platform application data types onto the tag contents may require additional attention to perform it in a deterministic way; see Section 3.2 of [I-D.bormann-cbor-det] for more explanation as well as examples. As the CDE would continually need to address additional issues raised by the registration of new tags, this specification recommends that new tag registrations address deterministic encoding in the context of CDE. Note that not in all cases the tag's deterministic encoding constraints will be confined to its definition of Preferred Serialization.

A particularly difficult field to obtain deterministic encoding for is floating point numbers, partially because they themselves are often obtained from processes that are not entirely deterministic between platforms. See Section 3.2.2 of [I-D.bormann-cbor-det] for more details. Section 4.2.2 of RFC 8949 [STD94] presents a number of choices that need to be made to obtain the CBOR Common Deterministic Encoding (CDE). Here, CDE entirely recurs to Preferred Serialization and does not itself define any additional constraints.

However, this BCP responds to a perceived need to clarify some of the Preferred Serialization constraints. Specifically, CDE specifies (in the order of the bullet list at the end of Section 4.2.2 of RFC 8949 [STD94]):

2. Besides the mandated use of Preferred Serialization, there is no further specific action for the two different zero values, e.g., an encoder that is asked by an application to represent a negative floating point zero will generate 0xf98000.
3. There is no attempt to mix integers and floating point numbers, i.e., all floating point values are encoded as the preferred floating-point representation that accurately represents the value, independent of whether the floating point value is, mathematically, an integral value (choice 2 of the second bullet).

4. Apart from finite and infinite numbers, [IEEE754] floating point values include NaN (not a number) values [I-D.bormann-cbor-numbers]. In CDE, there is no special handling of NaN values, except a clarification that the Preferred Serialization rules also apply to NaNs (with zero or non-zero payloads), using the encoding of NaNs as defined in Section 6.2.1 of [IEEE754]. Specifically, this means that shorter forms of encodings for a NaN are used when that can be achieved by only removing trailing zeros in the NaN payload (example serializations are available in Appendix A.1.2 of [I-D.bormann-cbor-numbers]). Further clarifying a "should"-level statement in Section 6.2.1 of [IEEE754], the CBOR encoding always uses a leading bit of 1 in the significand to encode a quiet NaN; the use of signaling NaNs by application protocols is NOT RECOMMENDED but when presented by an application these are encoded by using a leading bit of 0.

Typically, most applications that employ NaNs in their storage and communication interfaces will only use a single NaN value, quiet, non-negative NaN with payload 0, which therefore deterministically encodes as 0xf97e00.

5. There is no special handling of subnormal values.
6. CDE does not presume equivalence of basic floating point values with floating point values using other representations (e.g., tag 4/5). Such equivalences and related deterministic representation rules can be added at the ALDR level if desired, e.g., by stipulating additional equivalences and deterministically choosing exactly one representation for each such equivalence, and by restricting in general the set of data item values actually used by an application.

The main intent here is to preserve the basic generic data model, so applications (in their ALDR rules or by referencing a separate ALDR ruleset document, see Appendix B) can make their own decisions within that data model. E.g., an application's ALDR rules can decide that it only ever allows a single NaN value that would be encoded as 0xf97e00, so a CDE implementation focusing on this application would not even need to provide processing for other NaN values. Basing the definition of both CDE and ALDR rules on the generic data model of CBOR also means that there is no effect on the Concise Data Definition Language (CDDL) [RFC8610], except where the data description is documenting specific encoding decisions for byte strings that carry embedded CBOR (see Section 4).

3.2. Additional CDE Constraint from Basic Serialization

In addition to the encoding constraints from Preferred Serialization, CDE adds the constraint of not using indefinite length encoding from Basic Serialization. In many encoders, the use of indefinite length encoding is controlled by its configuration and can simply be switched off.

Some encoders turn to indefinite length encoding for arrays and maps with 256 or more elements/entries, to use the slightly smaller serialization size indefinite length encoding offers for these cases. Since leaving out support for indefinite length encoding is a common form of partial implementation, this may reduce interoperability. (Indefinite length encoding may also be used conditionally to avoid having to compute the total size ahead of time if the platform uses some form of chunking.) As CDE uses definite length encoding exclusively, such behavior needs to be turned off for CDE.

3.3. Additional CDE Constraint from CDE Itself

In line with Section 4.2.1 of RFC 8949 [STD94], CDE adds the constraint of map ordering to those from Basic Serialization. In some implementations, where platform representations of maps preserve ordering, this can be achieved using a generic CBOR encoder by pre-ordering all maps to be encoded, as long as that generic encoder also preserves the ordering in maps. In implementations without these properties, a specialized CBOR encoder may need to be employed.

Map ordering is a deterministic encoding constraint specific to maps, major type 5, that goes beyond maps' constraints for preferred serialization. In the definition of a tag being employed, there may also be some deterministic encoding constraints that are not covered by the tag's constraints for Preferred Serialization (see Section 3.1).

4. CDDL support

CDDL defines the structure of CBOR data items at the data model level; it enables being specific about the data items allowed in a particular place. It does not specify encoding, but CBOR protocols can specify the use of CDE (or simply Basic Serialization). For instance, it allows the specification of a floating point data item as "float16"; this means the application data model only foresees data that can be encoded as [IEEE754] binary16. Note that specifying "float32" for a floating point data item enables all floating point values that can be represented as binary32; this includes values that can also be represented as binary16 and that will be so represented in Basic Serialization.

[RFC8610] defines control operators to indicate that the contents of a byte string carries a CBOR-encoded data item (.cbor) or a sequence of CBOR-encoded data items (.cborseq).

CDDL specifications may want to specify that the data items should be encoded in Common CBOR Deterministic Encoding. The present specification adds two CDDL control operators that can be used for this.

The control operators .cde and .cdeseq are exactly like .cbor and .cborseq except that they also require the encoded data item(s) to be encoded according to CDE.

For example, a byte string of embedded CBOR that is to be encoded according to CDE can be formalized as:

```
leaf = #6.24(bytes .cde any)
```

More importantly, if the encoded data item also needs to have a specific structure, this can be expressed by the right-hand side (instead of using the most general CDDL type any here).

(Note that the .cdeseq control operator does not enable specifying different deterministic encoding requirements for the elements of the sequence. If a use case for such a feature becomes known, it could be added, or the CBOR sequence could be constructed with .join (Section 3.1 of [RFC9741]).)

Obviously, specifications that document ALDR rules can define related control operators that also embody the processing required by those ALDR rules, and are encouraged to do so.

5. Security Considerations

The security considerations in Section 10 of RFC 8949 [STD94] apply. The use of deterministic encoding can mitigate issues arising out of the use of non-preferred serializations specially crafted by an attacker. However, this effect only accrues if the decoder actually checks that deterministic encoding was applied correctly. More generally, additional security properties of deterministic encoding can rely on this check being performed properly.

6. IANA Considerations

```
// RFC Editor: please replace RFCXXXX with the RFC number of this RFC
// and remove this note.
```

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

Name	Reference
.cde	[RFCXXXX]
.cdeseq	[RFCXXXX]

Table 2: New control operators to be registered

7. References

7.1. Normative References

- [BCP14] Best Current Practice 14,
<<https://www.rfc-editor.org/info/bcp14>>.
At the time of writing, this BCP comprises the following:
- 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>>.
- 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>>.
- [IANA.cddl] IANA, "Concise Data Definition Language (CDDL)", <<https://www.iana.org/assignments/cddl>>.
- [IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2019, DOI 10.1109/IEEESTD.2019.8766229, <<https://ieeexplore.ieee.org/document/8766229>>.
- [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>>.
- [STD94] Internet Standard 94, <<https://www.rfc-editor.org/info/std94>>.

At the time of writing, this STD comprises the following:

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>>.

7.2. Informative References

[I-D.bormann-cbor-det]

Bormann, C., "CBOR: On Deterministic Encoding and Representation", Work in Progress, Internet-Draft, draft-bormann-cbor-det-04, 21 January 2025, <<https://datatracker.ietf.org/doc/html/draft-bormann-cbor-det-04>>.

[I-D.bormann-cbor-numbers]

Bormann, C., "On Numbers in CBOR", Work in Progress, Internet-Draft, draft-bormann-cbor-numbers-02, 7 July 2025, <<https://datatracker.ietf.org/doc/html/draft-bormann-cbor-numbers-02>>.

[I-D.bormann-dispatch-modern-network-unicode]

Bormann, C., "Modern Network Unicode", Work in Progress, Internet-Draft, draft-bormann-dispatch-modern-network-unicode-06, 2 March 2025, <<https://datatracker.ietf.org/doc/html/draft-bormann-dispatch-modern-network-unicode-06>>.

[I-D.ietf-cbor-edn-literals]

Bormann, C., "CBOR Extended Diagnostic Notation (EDN)", Work in Progress, Internet-Draft, draft-ietf-cbor-edn-literals-17, 12 May 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-cbor-edn-literals-17>>.

[I-D.mcnally-deterministic-cbor]

McNally, W., Allen, C., Bormann, C., and L. Lundblade, "dCBOR: A Deterministic CBOR Application Profile", Work in Progress, Internet-Draft, draft-mcnally-deterministic-cbor-12, 7 February 2025, <<https://datatracker.ietf.org/doc/html/draft-mcnally-deterministic-cbor-12>>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/rfc/rfc7493>>.

- [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/rfc/rfc8392>>.
- [RFC9581] Bormann, C., Gamari, B., and H. Birkholz, "Concise Binary Object Representation (CBOR) Tags for Time, Duration, and Period", RFC 9581, DOI 10.17487/RFC9581, August 2024, <<https://www.rfc-editor.org/rfc/rfc9581>>.
- [RFC9679] Isobe, K., Tschofenig, H., and O. Steele, "CBOR Object Signing and Encryption (COSE) Key Thumbprint", RFC 9679, DOI 10.17487/RFC9679, December 2024, <<https://www.rfc-editor.org/rfc/rfc9679>>.
- [RFC9741] Bormann, C., "Concise Data Definition Language (CDDL): Additional Control Operators for the Conversion and Processing of Text", RFC 9741, DOI 10.17487/RFC9741, March 2025, <<https://www.rfc-editor.org/rfc/rfc9741>>.
- [STD96] Internet Standard 96,
<<https://www.rfc-editor.org/info/std96>>.
At the time of writing, this STD comprises the following:
- Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <<https://www.rfc-editor.org/info/rfc9052>>.
- Schaad, J., "CBOR Object Signing and Encryption (COSE): Countersignatures", STD 96, RFC 9338, DOI 10.17487/RFC9338, December 2022, <<https://www.rfc-editor.org/info/rfc9338>>.
- [UAX-15] "Unicode Normalization Forms", Unicode Standard Annex, <<https://unicode.org/reports/tr15/>>.

Appendix A. Information Model, Data Model and Serialization

This appendix is informative.

For a good understanding of this document, it is helpful to understand the difference between an information model, a data model and serialization.

	Abstraction Level	Example	Standards	Implementation Representation
Information Model	Top level; conceptual	The temperature of something		
Data Model	Realization of information in data structures and data types	A floating- point number representing the temperature	CDDL	API input to CBOR encoder library, output from CBOR decoder library
Serialization	Actual bytes encoded for transmission	Encoded CBOR of a floating- point number	CBOR	Encoded CBOR in memory or for transmission

Table 3: A three-layer model of information representation

CBOR does not provide facilities for expressing information models. They are mentioned here for completeness and to provide some context.

CBOR defines a palette of basic data items that can be grouped into data types such as the usual integer or floating-point numbers, text or byte strings, arrays and maps, and certain special "simple values" such as Booleans and null. Extended data types may be constructed from these basic types. These basic and extended types are used to construct the data model of a CBOR protocol. One notation that is often used for describing the data model of a CBOR protocol is CDDL [RFC8610]. The various types of data items in the data model are serialized per RFC 8949 [STD94] to create encoded CBOR data items.

A.1. Data Model, Encoding Variants and Interoperability with Partial Implementations

In contrast to JSON, CBOR-related documents explicitly discuss the data model separately from its serialization. Both JSON and CBOR allow variation in the way some data items can be serialized:

- * In JSON, the number 1 can be serialized in several different ways (1, 0.1e1, 1.0, 100e-2) — while it may seem obvious to use 1 for this case, this is less clear for 10000000000000000000000000000000 vs. 1e+30 or 1e30. (As its serialization also doubles as a human-

readable interface, JSON also allows the introduction of blank space for readability.) The lack of an agreed data model for JSON led to the need for a complementary specification documenting an interoperable subset [RFC7493].

- * The CBOR standard addresses constrained environments, both by being concise and by limiting variation, but also by conversely allowing certain data items in the data model to be serialized in multiple ways, which may ease implementation on low-resource platforms. On the other hand, constrained environments may further save resources by only partially implementing the decoder functionality, e.g., by not implementing all those variations.

To deal with this encoding variation provided for certain data items, CBOR defines a _Preferred Serialization_ (Section 4.1 of RFC 8949 [STD94]). _Partial CBOR implementations_ are more likely to interoperate if their encoder uses Preferred Serialization and the decoder implements decoding at least the Preferred Serialization as well. A specific protocol for a constrained application may specify restrictions that allow, e.g., some fields to be of fixed length, guaranteeing interoperability even with partial implementations optimized for this application.

Another encoding variation is provided by indefinite-length encoding for strings, arrays, and maps, which enables these to be streamed without knowing their length upfront (Section 3.2 of RFC 8949 [STD94]). For applications that do not perform streaming of this kind, variation can be reduced (and often performance improved) by only allowing definite-length encoding. The present document coins the term _Basic Serialization_ for combining definite-length-only with preferred encoding, further reducing the variation that a decoder needs to deal with. The Common Deterministic Encoding, CDE, finally combines basic serialization with a deterministic ordering of entries in a map (Table 1).

Partial implementations of a representation format are quite common in embedded applications. Protocols for embedded applications often reduce the footprint of an embedded JSON implementation by explicitly restricting the breadth of the data model, e.g., by not using floating point numbers with 64 bits of precision or by not using floating point numbers at all. These data-model-level restrictions do not get in the way of using complete implementations ("generic encoders/decoders", Section 5.2 of RFC 8949 [STD94]). (Note that applications may need to complement deterministic encoding with decisions on the deterministic representation of application data into CBOR data items, see Appendix B.)

The increasing constraints on encoding (unconstrained, preferred, basic, CDE) are orthogonal to data-model-level data definitions as provided by [RFC8610]. To be useful in all applications, these constraints have been defined for all possible data items, covering the full range of values offered by CBOR's data types. This ensures that these serialization constraints can be applied to any CBOR protocol, without requiring protocol-specific modifications to generic encoder/decoder implementations.

Appendix B. Application-level Deterministic Representation

This appendix is informative.

CBOR application protocols are agreements about how to use CBOR for a specific application or set of applications.

For a CBOR protocol to provide deterministic representation, both the encoding and application layer must be deterministic. While CDE ensures determinism at the encoding layer, requirements at the application layer may also be necessary.

Application protocols make representation decisions in order to constrain the variety of ways in which some aspect of the information model could be represented in the CBOR data model for the application. For instance, there are several CBOR tags that can be used to represent a time stamp (such as tag 0, 1, 1001), each with some specific properties.

For example, an application protocol that needs to represent birthdate/times could specify:

- * At the sender's convenience, the birthdate/time MAY be sent either in epoch date format (as in tag 1) or string date format (as in tag 0).
- * The receiver MUST decode both formats.

While this specification is interoperable, it lacks determinism. There is variability in the application layer akin to variability in the CBOR encoding layer when CDE is not required.

To make this example application layer specification deterministic, allow only one date format (or at least be deterministic when there is a choice, e.g., by specifying string format for leap seconds only).

Application protocols that need to represent a timestamp typically choose a specific tag and further constrain its use where necessary (e.g., tag 1001 was designed to cover a wide variety of applications [RFC9581]). Where no tag is available, the application protocol can design its own format for some application data. Even where a tag is available, the application data can choose to use its definitions without actually encoding the tag (e.g., by using its content in specific places in an "unwrapped" form).

Another source of application layer variability comes from the variety of number types CBOR offers. For instance, the number 2 can be represented as an integer, float, big number, decimal fraction and other. Most protocols designs will just specify one number type to use, and that will give determinism, but here's an example specification that doesn't:

For instance, CWT [RFC8392] defines an application data type "NumericDate" which (as an application-level rule) is formed by "unwrapping" tag 1 (see Sections 2 and 5 of [RFC8392]). CWT does stop short of using deterministic encoding. A hypothetical deterministic variant of CWT would need to make an additional ALDR rule for NumericDate, as the definition of tag 1 allows both integer and floating point numbers (Section 3.4.2 of RFC 8949 [STD94]), which allows multiple application-level representations of integral numbers. These application rules may choose to only ever use integers, or to always use integers when the numeric value can be represented as such without loss of information, or to always use floating point numbers, or some of these for some application data and different ones for other application data.

Applications that require Deterministic Representation, and that derive CBOR data items from application data without maintaining a record of which choices are to be made when representing these application data, generally make rules for these choices as part of the application protocol. In this document, we speak about these choices as Application-level Deterministic Representation Rules (ALDR rules for short).

As an example, [RFC9679] is intended to derive a (deterministic) thumbprint from a COSE key [STD96]. Section 4 of [RFC9679] provides the rules that are used to construct a deterministic application-level representation (ALDR rules). Only certain data from a COSE key are selected to be included in that ALDR, and, where the COSE can choose multiple representations of semantically equivalent application data, the ALDR rules choose one of them, potentially requiring a conversion (Section 4.2 of [RFC9679]):

Note: [RFC9052] supports both compressed and uncompressed point representations. For interoperability, implementations adhering to this specification MUST use the uncompressed point representation. Therefore, the y-coordinate is expressed as a bstr. If an implementation uses the compressed point representation, it MUST first convert it to the uncompressed form for the purpose of thumbprint calculation.

CDE provides for encoding commonality between different applications of CBOR once these application-level choices have been made. It can be useful for an application or a group of applications to document their choices aimed at deterministic representation of application data in a general way, constraining the set of data items handled (exclusions, e.g., no compressed point representations) and defining further mappings (reductions, e.g., conversions to uncompressed form) that help the application(s) get by with the exclusions. This can be done in the application protocol specification (as in [RFC9679]) or as a separate document.

An early example of a separate document is the dCBOR specification [I-D.mcnally-deterministic-cbor]. dCBOR specifies the use of CDE together with some application-level rules, i.e., an ALDR ruleset, such as a requirement for all text strings to be in Unicode Normalization Form C (NFC) [UAX-15] — this specific requirement is an example for an exclusion of non-NFC data at the application level, and it invites implementing a reduction by routine normalization of text strings.

ALDR rules (including rules specified in a ALDR ruleset document) enable simply using implementations of the common CDE; they do not "fork" CBOR in the sense of requiring distinct generic encoder/decoder implementations for each application.

An implementation of specific ALDR rules combined with a CDE implementation produces well-formed, deterministically encoded CBOR according to [STD94], and existing generic CBOR decoders will therefore be able to decode it, including those that check for Deterministic Encoding ("CDE-checking decoders", see also Appendix C). Similarly, generic CBOR encoders will be able to produce valid CBOR that can be ingested by an implementation that enforces an application's ALDR rules if the encoder was handed data model level information from an application that simply conformed to those ALDR rules.

Please note that the separation between standard CBOR processing and the processing required by the ALDR rules is a conceptual one: Instead of employing generic encoders/decoders, both ALDR rule processing and standard CBOR processing can be combined into a specialized encoder/decoder specifically designed for a particular set of ALDR rules.

ALDR rules are intended to be used in conjunction with an application, which typically will naturally use a subset of the CBOR generic data model, which in turn influences which subset of the ALDR rules is used by the specific application (in particular if the application simply references a more general ALDR ruleset document). As a result, ALDR rules themselves place no direct requirement on what minimum subset of CBOR is implemented. For instance, a set of ALDR rules might include rules for the processing of floating point values, but there is no requirement that implementations of that set of ALDR rules support floating point numbers (or any other kind of number, such as arbitrary precision integers or 64-bit negative integers) when they are used with applications that do not use them.

Appendix C. Implementers' Checklists

This appendix is informative. It provides brief checklists that implementers can use to check their implementations. It uses RFC2119 language, specifically the keyword MUST, to highlight the specific items that implementers may want to check. It does not contain any normative mandates. This appendix is informative.

Notes:

- * This is largely a restatement of parts of Section 4 of RFC 8949 [STD94]. The purpose of the restatement is to aid the work of implementers, not to redefine anything.

Preferred Serialization Encoders as well as CDE Encoders and CDE-checking Decoders have certain properties that are expressed using RFC2119 keywords in this appendix.

- * Duplicate map keys are never valid in CBOR at all (see list item "Major type 5" in Section 3.1 of RFC 8949 [STD94]) no matter what sort of serialization is used. Of the various strategies listed in Section 5.6 of RFC 8949 [STD94], detecting duplicates and handling them as an error instead of passing invalid data to the application is the most robust one; achieving this level of robustness is a mark of quality of implementation.

- * Preferred serialization and CDE only affect serialization. They do not place any requirements, exclusions, mappings or such on the data model level. ALDR rules such as the ALDR ruleset defined by dCBOR are different as they can affect the data model by restricting some values and ranges.
- * CBOR decoders in general (as opposed to "CDE-checking decoders" specifically advertised as supporting CDE) are not required to check for preferred serialization or CDE and reject inputs that do not fulfill these requirements. However, in an environment that employs deterministic encoding, employing non-checking CBOR decoders negates many of its benefits. Decoder implementations that advertise "support" for preferred serialization or CDE need to check the encoding and reject input that is not encoded to the encoding specification in use. Again, ALDR rules such as those in dCBOR may pose additional requirements, such as requiring rejection of non-conforming inputs.

If a generic decoder needs to be used that does not "support" CDE, a simple (but somewhat clumsy) way to check its input for proper CDE encoding is to re-encode the decoded data with CDE and check for bit-to-bit equality with the original input.

C.1. Preferred Serialization

In the following, the abbreviation "ai" will be used for the 5-bit additional information field in the first byte of an encoded CBOR data item, which follows the 3-bit field for the major type.

C.1.1. Preferred Serialization Encoders

1. Shortest-form encoding of the argument **MUST** be used for all major types. Major type 7 is used for floating-point and simple values; floating point values have its specific rules for how the shortest form is derived for the argument. The shortest form encoding for any argument that is not a floating point value is:
 - * 0 to 23 and -1 to -24 **MUST** be encoded in the same byte as the major type.
 - * 24 to 255 and -25 to -256 **MUST** be encoded only with an additional byte (ai = 0x18).
 - * 256 to 65535 and -257 to -65536 **MUST** be encoded only with an additional two bytes (ai = 0x19).
 - * 65536 to 4294967295 and -65537 to -4294967296 **MUST** be encoded only with an additional four bytes (ai = 0x1a).

2. If floating-point numbers are emitted, the following apply:

- * The length of the argument indicates half (binary16, ai = 0x19), single (binary32, ai = 0x1a) and double (binary64, ai = 0x1b) precision encoding. If multiple of these encodings preserve the precision of the value to be encoded, only the shortest form of these MUST be emitted. That is, encoders MUST support half-precision and single-precision floating point.
- * [IEEE754] Infinities and NaNs, and thus NaN payloads, MUST be supported, to the extent possible on the platform.

As with all floating point numbers, Infinities and NaNs MUST be encoded in the shortest of double, single or half precision that preserves the value:

- Positive and negative infinity and zero MUST be represented in half-precision floating point.
- For NaNs, the value to be preserved includes the sign bit, the quiet bit, and the NaN payload (whether zero or non-zero). The shortest form is obtained by removing the rightmost N bits of the payload, where N is the difference in the number of bits in the significand (mantissa representation) between the original format and the shortest format. This trimming is performed only (preserves the value only) if all the rightmost bits removed are zero. (This means that a double or single quiet NaN that has a zero NaN payload will always be represented in a half-precision quiet NaN.)

3. If tags 2 and 3 are supported, the following apply:

- * Positive integers from 0 to $2^{64} - 1$ MUST be encoded as a type 0 integer.
- * Negative integers from $-(2^{64})$ to -1 MUST be encoded as a type 1 integer.
- * Leading zeros MUST NOT be present in the byte string content of tag 2 and 3.

(This also applies to the use of tags 2 and 3 within other tags, such as 4 or 5.)

C.1.2. Decoders and Preferred Serialization

There are no special requirements that CBOR decoders need to meet to be what could be called a "Preferred Serialization Decoder".

Partial decoder implementations that want to accept at least Preferred Serialization need to pay attention to at least the following requirements:

1. Decoders **MUST** accept shortest-form encoded arguments (see Section 3 of RFC 8949 [STD94]).
2. If arrays or maps are supported, both definite-length and indefinite-length arrays or maps **MUST** be accepted.
3. If text or byte strings are supported, both definite-length and indefinite-length text or byte strings **MUST** be accepted.
4. If floating-point numbers are supported, the following apply:
 - * Half-precision values **MUST** be accepted.
 - * Double- and single-precision values **SHOULD** be accepted; leaving these out is only foreseen for decoders that need to work in exceptionally constrained environments.
 - * If double-precision values are accepted, single-precision values **MUST** be accepted.
 - * Infinites and NaNs, and thus NaN payloads, **MUST** be accepted and presented to the application (not necessarily in the platform number format, if that doesn't support those values).
5. If big numbers (tags 2 and 3) are supported, type 0 and type 1 integers **MUST** be accepted where a tag 2 or 3 would be accepted. Leading zero bytes in the tag content of a tag 2 or 3 **MUST** be ignored.

C.2. Basic Serialization

Basic Serialization further restricts Preferred Serialization by not using indefinite length encoding. A CBOR encoder can choose to employ Basic Serialization in order to reduce the variability that needs to be handled by decoders, potentially maximizing interoperability with partial (e.g., constrained) CBOR decoder implementations.

C.2.1. Basic Serialization Encoders

The Basic Serialization Encoder requirements are identical to the Preferred Serialization Encoder requirements, with the following additions:

1. If maps or arrays are emitted, they MUST use definite-length encoding (never indefinite-length).
2. If text or byte strings are emitted, they MUST use definite-length encoding (never indefinite-length).

C.2.2. Decoders and Basic Serialization

There are no special requirements that CBOR decoders need to meet to be what could be called a "Basic Serialization Decoder".

Partial decoder implementations that want to accept at least Basic Serialization need to pay attention to the requirements for partial decoder implementations that accept Preferred Serialization, with the following relaxations from the items 2 and 3 of Appendix C.1.2:

1. If arrays or maps are supported, definite-length arrays or maps MUST be accepted.
2. If text or byte strings are supported, definite-length text or byte strings MUST be accepted.

C.3. CDE

C.3.1. CDE Encoders

1. CDE encoders MUST only emit CBOR that fulfills the basic serialization rules (Appendix C.2.1).
2. CDE encoders MUST sort maps by the CBOR representation of the map key. The sorting is byte-wise lexicographic order of the encoded map key data items.
3. CDE encoders MUST generate CBOR that fulfills basic validity (Section 5.3.1 of RFC 8949 [STD94]). Note that this includes not emitting duplicate keys in a major type 5 map as well as emitting only valid UTF-8 in major type 3 text strings.

Note also that CDE does NOT include a requirement for Unicode normalization [UAX-15]; Appendix C of [I-D.bormann-dispatch-modern-network-unicode] contains some rationale that went into not requiring routine use of Unicode normalization processes.

C.3.2. CDE-checking Decoders

The term "CDE-checking Decoder" is a shorthand for a CBOR decoder that advertises `_supporting_` CDE (see the start of this appendix).

1. CDE-checking decoders MUST follow the rules for decoders that accept Basic Serialization (Appendix C.2.2) and MUST check the input for keeping the Basic Serialization constraints.
2. CDE-checking decoders MUST check for ordering map keys and for basic validity of the CBOR encoding (see Section 5.3.1 of RFC 8949 [STD94], which includes a check against duplicate map keys and invalid UTF-8).

To be called a CDE-checking decoder, it MUST NOT present to the application a decoded data item that fails one of these checks (except maybe via special diagnostic channels with no potential for confusion with a correctly CDE-decoded data item).

Appendix D. Encoding Examples

The following three tables provide examples of CDE-encoded CBOR data items, each giving Diagnostic Notation (EDN [I-D.ietf-cbor-edn-literals]), the encoded data item in hexadecimal, and a comment.

Implementers that want to use these examples as test input may be interested in the file `example-table-input.csv` in the github repository `cbor-wg/draft-ietf-cbor-cde`.

D.1. Integer Value Examples

EDN	CBOR (hex)	Comment
0	00	Smallest unsigned immediate int
-1	20	Largest negative immediate int
23	17	Largest

-24	37	unsigned immediate int Smallest negative
24	1818	immediate int Smallest unsigned one- byte int
-25	3818	Largest negative one- byte int
255	18ff	Largest unsigned one- byte int
-256	38ff	Smallest negative one- byte int
256	190100	Smallest unsigned two- byte int
-257	390100	Largest negative two- byte int
65535	19ffff	Largest unsigned two- byte int
-65536	39ffff	Smallest negative two- byte int
65536	1a00010000	Smallest unsigned four- byte int
-65537	3a00010000	Largest negative four- byte int
4294967295	1affffffff	Largest unsigned four- byte int
-4294967296	3affffffff	Smallest negative four- byte int
4294967296	1b0000000100000000	Smallest unsigned eight-byte int
-4294967297	3b0000000100000000	Largest negative eight-byte int
18446744073709551615	1bffffffffffffffff	Largest

-18446744073709551616	3bffffffffffffffffffff	unsigned eight-byte int Smallest negative eight-byte int
18446744073709551616	c249010000000000000000	Smallest unsigned bigint
-18446744073709551617	c349010000000000000000	Largest negative bigint

Table 4: Integer Value Examples

D.2. Floating Point Value Examples

EDN	CBOR (hex)	Comment
0.0	f90000	Zero
-0.0	f98000	Negative zero
Infinity	f97c00	Infinity
-Infinity	f9fc00	-Infinity
NaN	f97e00	NaN
NaN	f97e01	NaN with non-zero payload
5.960464477539063e-8	f90001	Smallest positive 16-bit float (subnormal)
0.00006097555160522461	f903ff	Largest positive subnormal 16-bit float
0.00006103515625	f90400	Smallest non-subnormal positive 16-bit float
65504.0	f97bff	Largest positive 16-bit float
1.401298464324817e-45	fa00000001	Smallest positive 32-bit float (subnormal)
1.1754942106924411e-38	fa007fffff	Largest positive

1.1754943508222875e-38	fa00800000	subnormal 32-bit float Smallest non- subnormal positive 32-bit float
3.4028234663852886e+38	fa7f7fffff	Largest positive 32-bit float
5.0e-324	fb0000000000000001	Smallest positive 64-bit float (subnormal)
2.225073858507201e-308	fb000fffffffffffffff	Largest positive subnormal 64-bit float
2.2250738585072014e-308	fb0010000000000000	Smallest non- subnormal positive 64-bit float
1.7976931348623157e+308	fb7fefffffffffffffff	Largest positive 64-bit float
-0.000003333333333333333333	fbbecbf647612f3696	Arbitrarily selected number
10.559998512268066	fa4128f5c1	- "-
10.559998512268068	fb40251eb820000001	Next in succession
295147905179352830000.0	fa61800000	2^68 (diagnostic notation truncates precision)
2.0	f94000	Number without a fractional part
-5.960464477539063e-8	f98001	Largest negative subnormal
-5.960464477539062e-8	fbbe6fffffffffffffff	16-bit float Adjacent to largest negative subnormal 16-bit float

-5.960464477539064e-8	fbbe70000000000001	-"
-5.960465188081798e-8	fab3800001	-"
0.0000609755516052246	fb3f0ff7ffffffffffff	Adjacent to largest subnormal 16-bit float
0.000060975551605224616	fb3f0ff80000000001	-"
0.00006097555243203416	fa387fc001	-"
0.00006103515624999999	fb3f0fffffffffffffff	Adjacent to smallest 16-bit float
0.00006103515625000001	fb3f10000000000001	-"
0.00006103516352595761	fa38800001	-"
65503.999999999999	fb40effbffffffffffff	Adjacent to largest 16-bit float
65504.000000000001	fb40effc0000000001	-"
65504.00390625	fa477fe001	-"
1.4012984643248169e-45	fb369fffffffffffffff	Adjacent to smallest subnormal 32-bit float
1.4012984643248174e-45	fb36a0000000000001	-"
1.175494210692441e-38	fb380ffffffbffffffff	Adjacent to largest subnormal 32-bit float
1.1754942106924412e-38	fb380fffffc0000001	-"
1.1754943508222874e-38	fb380fffffffffffffff	Adjacent to smallest 32-bit float
1.1754943508222878e-38	fb3810000000000001	-"
3.4028234663852882e+38	fb47efffffdfffffffff	Adjacent to largest 32-bit float
3.402823466385289e+38	fb47efffffe0000001	-"

Table 5: Floating Point Value Examples

D.3. Failing Examples

EDN	CBOR (hex)	Comment
{"b":0,"a":1}	a2616200616101	Incorrect map key ordering

[4, 5]	98020405	Array length not in preferred encoding
255	1900ff	Integer not in preferred encoding
-18446744073709551617	c34a00010000000000000000	Bigint with leading zero bytes
10.5	fa41280000	Not in preferred encoding
NaN	fa7fc00000	Not in preferred encoding
65536	c243010000	Integer value too small for bigint
(_ h'01', h'0203')	5f4101420203ff	Indefinite length encoding
(Not CBOR)	f818	Simple values 24..31 not in use
(Not CBOR)	fc	Reserved (ai = 28..30)

Table 6: Failing Examples

Appendix E. Examples for Preferred Serialization of Integers

This appendix looks at the set of encoded CBOR data items that represent the integer number 1. Preferred Serialization chooses one of them (0x01), which is then always used to encode the number. The CDE encoding constraints include those of preferred serialization. A CDE-checking decoder checks that no other serialization is being used in the encoded data item being decoded.

Serialization of integer number 1	Preferred?
0x01	yes (shortest mt0)
0x1801, 0x190001, 0x1a00000001, 0x1b0000000000000001	no (mt0, but not shortest argument)
0xc24101	no (could use mt0)
0xc2420001, 0xc243000001, etc.	no (could use mt0, uses leading zeros)
0xc25f41004101ff, and similar	no (could use mt0, uses leading zeros)

Table 7: Serializations of integer number 1

For the integer number 10000000000000000000 (1 with 20 decimal zeros), the only `_basic_` serialization is:

```
C2          # tag(2)
  49        # bytes(9)
    056BC75E2D63100000 #
```

(Note that, in addition to this serialization, there are multiple serializations that would also count as `_preferred_` serializations, as the preferred serialization constraint by itself does not exclude indefinite length encoding of the byte string that is the content of tag 2.)

Appendix F. Example Code for Encoding into 16-bit Floating Point

Appendix D (Half-Precision) of RFC 8949 [STD94] provides example C and Python code for decoding 16-bit ("Half Precision", binary16) floating point numbers. Providing this code was considered important at the time to aid in the creation of generic decoders.

Given that CDE implementations that support floating point Numbers not only need to decode, but also to encode their 16-bit format, this appendix provides example C code to convert a floating point number that is in 64-bit form ("Double Precision", binary64) into binary16.

If such a conversion is not possible (i.e., there is no 16-bit representation for the 64-bit value given), the function `try_float16_encode` returns -1. Otherwise it returns a two-byte integer (range 0x0000 to 0xFFFF) that, prefixed with 0xF9, is suitable to encode the value.

```
/* returns 0..0xFFFF if float16 encoding possible, -1 otherwise.
   b64 is a binary64 floating point as an unsigned long. */
int try_float16_encode(unsigned long b64) {
    unsigned long s16 = b64 >> 48 & 0x8000;
    unsigned long mant = b64 & 0xffffffffffffUL;
    unsigned long exp = b64 >> 52 & 0x7fff;
    if (exp == 0 && mant == 0) /* f64 denorms are out of range */
        return s16; /* so handle 0.0 and -0.0 only */
    if (exp >= 999 && exp < 1009) { /* f16 denorm, exp16 = 0 */
        if (mant & ((1UL << (1051 - exp)) - 1))
            return -1; /* bits lost in f16 denorm */
        return s16 + ((mant + 0x100000000000000UL) >> (1051 - exp));
    }
    if (mant & 0x3fffffffffffUL) /* bits lost in f16 */
        return -1;
    if (exp >= 1009 && exp <= 1038) /* normalized f16 */
        return s16 + ((exp - 1008) << 10) + (mant >> 42);
    if (exp == 2047) /* Inf, NaN */
        return s16 + 0x7c00 + (mant >> 42);
    return -1;
}
```

Figure 1: Example C Code for a Half-Precision Encoder

List of Figures

Figure 1: Example C Code for a Half-Precision Encoder

List of Tables

Table 1:	Constraints on the Serialization of CBOR
Table 2:	New control operators to be registered
Table 3:	A three-layer model of information representation
Table 4:	Integer Value Examples
Table 5:	Floating Point Value Examples
Table 6:	Failing Examples
Table 7:	Serializations of integer number 1

Acknowledgments

An early version of this document was based on the work of Wolf McNally and Christopher Allen as documented in [I-D.mcnally-deterministic-cbor], which serves as an example for an ALDR ruleset document. We would like to explicitly acknowledge that this work has contributed greatly to shaping the concept of a CBOR Common Deterministic Encoding and the use of ALDR rules/rulesets on top of that. Mikolai Gtschow proposed adding Section 2. Anders Rundgren provided most of the initial text that turned into Appendix D.

Contributors

Laurence Lundblade
Security Theory LLC
Email: lgl@securitytheory.com

Laurence provided most of the text that became Appendix A and Appendix C.

Author's Address

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