Workgroup: CBOR Working Group
Internet-Draft: draft-ietf-cbor-file-magic-07
Published: 15 December 2021
Intended Status: Best Current Practice
Expires: 18 June 2022
Authors: M. Richardson
Sandelman Software Works
C. Bormann
Universität Bremen TZI

On storing CBOR encoded items on stable storage

Abstract

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

About This Document

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

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

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Since very early in computing, operating systems have sought ways to mark which files could be processed by which programs.

For instance, the Unix file(1) command, which has existed since 1973 [file], has been able to identify many file formats for decades based upon the contents of the file.

Many systems (Linux, macOS, Windows) will select the correct application based upon the file contents, if the system can not determine it by other means. For instance, starting in MacOS, a resource fork was maintained that includes media type ("MIME type") information and therefore ideally never needs to know anything about the file.

But, many other systems do this by file extensions. Many common web servers derive the MIME-type information from file extensions.

While having a media type associated with the file is a better solution in general, when files become disconnected from their type information, such as when attempting to do forensics on a damaged system, then being able to identify a file type can become very important.

It is noted that in the media type registration, that a magic number is asked for, if available, as is a file extension.

A challenge for the file(1) program is often that it can be confused by the encoding vs the content. For instance, an Android "apk" used to transfer and store an application may be identified as a ZIP file. Additionally, both OpenOffice and MSOffice files are ZIP files of XML files.

As CBOR becomes a more and more common encoding for a wide variety of artifacts, identifying them as just "CBOR" is probably not sufficient. This document provides a way to encode a magic number into the beginning of a CBOR format file. Two possible methods of enveloping data are presented: a CBOR Protocol author will specify one. (A CBOR Protocol is a specification which uses CBOR as its encoding.)

Examples of CBOR Protocols currently under development include CoSWID [I-D.ietf-sacm-coswid], and EAT [I-D.ietf-rats-eat]. COSE itself [RFC8152] is considered infrastructure, however the encoding of public keys in CBOR as described in [I-D.ietf-cose-cbor-encoded-cert] would be an identified CBOR Protocol as well.

A major inspiration for this document is observing the mess in certain ASN.1 based systems where most files are PEM encoded,

identified by the extension "pem", confusing public keys, private keys, certificate requests, and S/MIME content.

While these envelopes add information to how data conforming to CBOR Protocols are stored in files, there is no requirement that either type of envelope be transferred on the wire.

In addition to the on-disk identification aspects, there are some protocols which may benefit from having such a magic number on the wire if they are presently using a different (legacy) encoding scheme. The presence of the identifiable magic sequence signals that CBOR is being used as opposed to a legacy scheme. In addition, for convenience, Appendix C defines a simple way to retroactively add a magic number to content-formats as defined by [RFC7252], even if not in CBOR form.

1.1. Terminology

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

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

1.2. Requirements for a Magic Number

A magic number is ideally a fingerprint that is unique to a CBOR protocol, present in the first few (small multiple of 4) bytes of the file, which does not change when the contents change, and does not depend upon the length of the file.

Less ideal solutions have a pattern that needs to be matched, but in which some bytes need to be ignored. While the Unix file(1) command can be told to ignore certain bytes, this can lead to ambiguities.

2. Protocol

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

2.1. The CBOR Protocol Specific Tag

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

The IANA policy for 4-byte CBOR Tags is First Come First Served, so all that is required is an email to IANA, having filled in the small template provided in Section 9.2 of [RFC8949].

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

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

For CBOR byte strings that happen to contain a representation that is described by a CoAP Content-Format Number (Section 12.3 of [RFC7252], Registry CoAP Content-Formats of [IANA.core-parameters]), a tag number has already been allocated in Section 5.3 (see Appendix A for details and examples).

2.2. Enveloping Method: CBOR Tag Wrapped

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

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

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

This method wraps the CBOR value as tags usually do. Applications that need to send the CBOR value across a constrained link may wish to remove the two tags if the use is implicitly understood.

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

2.2.1. Example

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

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

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

Or in hex:

```
d9 d9f7                     # tag(55799)
   da 63740070              # tag(1668546672)
      81                    # array(1)
         a3                 # map(3)
            00              # unsigned(0)
            67              # text(7)
               63757272656e74 # "current"
            06              # unsigned(6)
            03              # unsigned(3)
            02              # unsigned(2)
            f9 3e00         # primitive(15872)
```

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

2.3. Enveloping Method: CBOR Tag Sequence

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

This method prepends a data item to the sequence to be tagged that consists of two tags nested around a constant string for a total of 12 bytes.

1. The file shall start with the Self-described CBOR Sequence tag, 55800.

2. The file shall continue with a CBOR tag, from the First Come First Served space, which uniquely identifies the CBOR Protocol. As with the previous method, the use of a four-byte tag is encouraged that encodes without zero bytes.

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

The first part identifies the file as being a CBOR Sequence, and does so with all the desirable properties explained in Section 3.4.6 of [RFC8949]. Specifically, it does not appear to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The second part identifies which CBOR Protocol is used, as described above.

The third part is represented as a constant byte sequence 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three byte sequence 0x42_4f_52 ('BOR' in diagnostic notation). This is the data item that is being tagged.

The actual CBOR Protocol value then follows as the next data item(s) in the CBOR sequence, without a need for any further specific tag. The use of a CBOR Sequence allows the application to trivially remove the first item with the two tags.

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

3. Advice to Protocol Developers

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

Which should a protocol designer use?

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

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

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

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

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

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

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

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

Here are some considerations in the form of survey questions:

3.1. Is the on-wire format new?

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

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

3.2. Can many items be trivially concatenated?

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

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

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

3.3. Are there tags at the start?

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

4. Security Considerations

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

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

5. IANA Considerations

Section 5.1 documents the allocation that was done for a CBOR tag to be used in a CBOR sequence to identify the sequence (an example for using this tag is found in Appendix B). Section 5.3 allocates a CBOR tag for each actual or potential CoAP Content-Format number (examples are in Appendix A).

5.1. CBOR Sequence Tag

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

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

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

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

This is not valid UTF-8: the first 0xd9 introduces a three-byte sequence in UTF-8, but the 0xd9 as the second value is not a valid second byte for UTF-8.

This is not valid UTF-16: the byte sequence 0xd9d9 (in either endian order) puts this value into the UTF-16 high-half zone, which would signal that this a 32-bit Unicode value. However, the following 16-bit big-endian value 0xf8.. is not a valid second sequence according to [RFC2781]. On a little-endian system, it would be necessary to examine the fourth byte to determine if it is valid. That next byte

is determined by the subsequent encoding, and Section 3.4.6 of [RFC8949] has already determined that no valid CBOR encodings result in valid UTF-16.

Data Item:
tagged byte string

Semantics:
indicates that the file contains CBOR Sequences

5.2. CBOR Header Tag

IANA is requested to allocate tag 55801 as the tag for the CBOR Tag Header Enveloping Method (Appendix C).

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

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

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

This is not valid UTF-8: the first 0xd9 introduces a three-byte sequence in UTF-8, but the 0xd9 as the second value is not a valid second byte for UTF-8.

This is not valid UTF-16: the byte sequence 0xd9d9 (in either endian order) puts this value into the UTF-16 high-half zone, which would signal that this a 32-bit Unicode value. However, the following 16-bit big-endian value 0xf9.. is not a valid second sequence according to [RFC2781]. On a little-endian system, it would be necessary to examine the fourth byte to determine if it is valid. That next byte is determined by the subsequent encoding, and Section 3.4.6 of [RFC8949] has already determined that no valid CBOR encodings result in valid UTF-16.

Data Item:
tagged byte string

Semantics:
indicates that the file starts with a CBOR Header

5.3. CBOR Tags for CoAP Content-Format Numbers

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

Data Item:
byte string

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

Reference:
RFCthis

6. References

6.1. Normative References

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/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/info/rfc8949>.

6.2. Informative References

[file] Wikipedia, "file (command)", 20 January 2021, <https://en.wikipedia.org/wiki/File_%28command%29>.

[I-D.ietf-core-new-block] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", Work in Progress, Internet-Draft, draft-ietf-core-new-block-14, 26 May 2021, <https://www.ietf.org/archive/id/draft-ietf-core-new-block-14.txt>.

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

[I-D.ietf-rats-eat] Lundblade, L., Mandyam, G., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-11, 24 October 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-11.txt>.

[I-D.ietf-sacm-coswid] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-

coswid-19, 20 October 2021, <https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-19.txt>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[RFC2781] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, DOI 10.17487/RFC2781, February 2000, <https://www.rfc-editor.org/info/rfc2781>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

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

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

Appendix A. CBOR Tags for CoAP Content Formats

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

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

This specification allocates CBOR tag numbers 1668546560 (0x63740000) to 1668612095 (0x6374FFFF) for the tagging of representations of specific content formats. The tag content tagged with tag number NNNNNNNN (in above range) is a byte string that is to be interpreted as a representation of the content format NNNNNNNN-1668546560. Exceptionally, when used immediately as tag content within a tag 58000 (Section 2.3) or 58001 (Appendix C), the tag content is the byte string 'BOR', signifying that the representation of the given content-format follows in the way defined for these tags.

A.1. Content-Format Tag Examples

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

* As discussed in Section 2.2.1, Content-Format 112 stands for media type application/senml+cbor (no parameters). The corresponding tag number is 1668546672 (i.e., 1668546560+112).

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

```
senml-cbor = #6.1668546672(bstr)
```

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

* Content-Format 272 stands for media type application/missing-blocks+cbor-seq, a CBOR sequence [I-D.ietf-core-new-block].

  The corresponding tag number is 1668546832 (i.e., 1668546560+272).

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

```
missing-blocks = #6.1668546832(bstr)
```

Appendix B. Example from Openswan

The Openswan IPsec project has a daemon ("pluto"), and two control programs ("addconn", and "whack"). They communicate via a Unix-domain socket, over which a C-structure containing pointers to strings is serialized using a bespoke mechanism. This is normally not a problem as the structure is compiled by the same compiler; but when there are upgrades it is possible for the daemon and the control programs to get out of sync by the bespoke serialization. As a result, there are extra compensations to deal with shutting the daemon down. During testing it is sometimes the case that upgrades are backed out.

In addition, when doing unit testing, the easiest way to load policy is to use the normal policy reading process, but that is not normally loaded in the daemon. Instead the IPC that is normally sent

across the wire is compiled/serialized and placed in a file. The above magic number is included in the file, and also on the IPC in order to distinguish the "shutdown" command CBOR operation.

In order to reduce the problems due to serialization, the serialization is being changed to CBOR. Additionally, this change allows the IPC to be described by CDDL, and for any language that encode to CBOR can be used.

IANA has allocated the tag 1330664270, or 0x4f_50_53_4e for this purpose. As a result, each file and each IPC is prefixed with a CBOR TAG Sequence.

In diagnostic notation:

```
55800(1330664270(h'424F52'))
```

Or in hex:

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

Appendix C. Using CBOR headers for non-CBOR data

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

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

1. The file shall start with the Self-described CBOR header tag, 55801.

2. The file shall continue with a CBOR tag, from the First Come First Served space, which uniquely identifies the non-CBOR Protocol. As with the previous method, the use of a four-byte tag is encouraged that encodes without zero bytes.

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

The first part identifies the file as being prefixed by a CBOR header, and does so with all the desirable properties explained in

Section 3.4.6 of [RFC8949]. Specifically, it does not appear to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The second part identifies which non-CBOR Protocol is used, as described above.

The third part is represented as a constant byte sequence 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three byte sequence 0x42_4f_52 ('BOR' in diagnostic notation). This is the data item that is being tagged.

The actual non-CBOR Protocol value then follows directly appended to the CBOR header data item. The use of a CBOR header allows the application to trivially remove the header item with the two tags.

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

C.1. Content-Format Tag Examples

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

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

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

  ```
  td-json-header = #6.55801(#6.1668546992('BOR'))
  ```

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

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

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

  ```
  json-deflate-header = #6.55801(#6.1668557610('BOR'))
  ```

Appendix D. Changelog

Acknowledgements

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

Contributors

Josef 'Jeff' Sipek

Email: jeffpc@josefsipek.net

Authors' Addresses

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

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

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