

Workgroup: CBOR Working Group  
Internet-Draft: draft-ietf-cbor-file-magic-03  
Published: 4 August 2021  
Intended Status: Best Current Practice  
Expires: 5 February 2022  
Authors: M. Richardson  
Sandelman Software Works

## **On storing CBOR encoded items on stable storage**

### **Abstract**

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

This document is being discussed at: <https://github.com/cbor-wg/cbor-magic-number>

### **Status of This Memo**

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

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

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

This Internet-Draft will expire on 5 February 2022.

### **Copyright Notice**

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

- [1. Introduction](#)
  - [1.1. Terminology](#)
- [2. Requirements for a Magic Number](#)
- [3. Protocol](#)
  - [3.1. The CBOR Protocol Specific Tag](#)
  - [3.2. CBOR Tag Wrapped](#)
  - [3.3. CBOR Tag Sequence](#)
- [4. Advice to Protocol Developers](#)
  - [4.1. Is the on-wire format new?](#)
  - [4.2. Can many items be trivially concatenated?](#)
  - [4.3. Are there tags at the start?](#)
- [5. Security Considerations](#)
- [6. IANA Considerations](#)
  - [6.1. CBOR Sequence Tag](#)
  - [6.2. CBOR Tags for CoAP Content-Format Numbers](#)
- [7. References](#)
  - [7.1. Normative References](#)
  - [7.2. Informative References](#)
- [Appendix A. CBOR Tags for CoAP Content Formats](#)
  - [A.1. Content-Format Tag Examples](#)
- [Appendix B. Example from Openswan](#)
- [Appendix C. Changelog](#)
- [Acknowledgements](#)
- [Contributors](#)
- [Author's Address](#)

## 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. 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, the classic MacOS maintained a resource fork that includes media type ("MIME type") information and therefore ideally never needs to know anything about the file. Other systems do this by 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. Both OpenOffice or MSOffice files are ZIP files of XML files. (Unless OpenOffice files are flat (fodp) files, in which case they may appear to be generic 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 options are presented: typically 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.

A major inspiration for this document is observing the mess in 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.

These proposals are invasive to how CBOR protocols are written to disk, but in both cases, the proposed envelope does not require that the tag 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 presently using a different (legacy) encoding scheme. The presence of the identifiable magic sequence signals that CBOR is being used or a legacy scheme.

### **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](#)].

## 2. Requirements for a Magic Number

A magic number is ideally a unique fingerprint, present in the first 4 or 8 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 bytes, this can lead to ambiguities.

## 3. Protocol

There are two variations of this practice. Both use CBOR Tags in a way that results in a deterministic first 8 to 12 bytes.

### 3.1. The CBOR Protocol Specific Tag

CBOR Protocol designers should obtain a 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 policy 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 should be allocated by the author of the CBOR Protocol, and to be in the four-byte range, it should be at least 0x01000000 (decimal 16777216) in value.

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](#)], Subregistry [Content-Formats](#) of [[IANA.core-parameters](#)]), a tag number has been allocated in [Section 6.2](#) (see [Appendix A](#) for details and examples).

### 3.2. CBOR Tag Wrapped

This proposal 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 proposal 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. This is a decision of the CBOR Protocol specification.

### 3.3. CBOR Tag Sequence

This proposal makes use of CBOR Sequences as described in [[RFC8742](#)].

This proposal consists of two tags and 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. The use of a four-byte tag is encouraged.
3. The 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 seem 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.

This means that should a 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.

## 4. Advice to Protocol Developers

This document introduces a choice between a CBOR Sequence and a wrapped CBOR Tag. 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 QRcode 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 is a little bit 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 an empty CBOR item. 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. But, when in binary DER format, concatenation of certificates was not compatible with most programs.

The use of CBOR Tag Wrapped format is easier to retrofit to an existing format with existing and unchangeable on-disk format. This new sequence of tags are expected to be trivially ignored by an existing program 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:

#### **4.1. 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 (and they usually are if CBOR is being considered), then the CBOR Tag Sequence format should be adopted for on-disk storage.

#### **4.2. Can many items be trivially concatenated?**

If the programs that read the contents of the file already expect to process all of the 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 the use of CBOR Tag Sequence may present an implementation hurdle to programs that previously just read a single value and used it.

#### **4.3. 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.

### **5. 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 corrupt them.

### **6. IANA Considerations**

[Section 6.1](#) 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 6.2](#) allocates a CBOR tag for each actual or potential CoAP Content-Format number (examples are in [Appendix A](#)).

## 6.1. CBOR Sequence Tag

IANA has allocated tag 55800 as the CBOR Sequence tag. 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 is: 0xd9\_d9\_f8.

This is not valid UTF-8: the first 0xd9 puts the value into the three-byte value of 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 a valid UTF-16.

**Data Item:**

byte string

**Semantics:**

indicates that the file contains CBOR Sequences

## 6.2. 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 NNNN, the representation of content-format (RFC7252) NNNN-1668546560

**Reference:**

RFCthis

## 7. References

### 7.1. 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>>.

## 7.2. Informative References

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

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

[I-D.ietf-rats-eat] Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-10, 7 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-rats-eat-10.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-18, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-18.txt>>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <<http://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 NNNN (in above range) is a byte string that is to be interpreted as a representation of the content format NNNN-1668546560.

### A.1. Content-Format Tag Examples

Subregistry [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 application/td+json representations:

```
td-json = #6.1668546992(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 11050 stands for media type application/json in deflate encoding.

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

So the following CDDL snippet can be used to identify application/json representations compressed in deflate encoding:

```
json-deflate = #6.1668557610(bstr)
```

The byte string is appropriate here as the type for the tag content, because the compressed form is an instance of a general byte string.

## 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:

In diagnostic notation:

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

Or in hex:

```
00000000 d9 d9 f8 da 4f 50 53 4e 43 42 4f 52 |...OPSNCBOR|
```

## **Appendix C. Changelog**

### **Acknowledgements**

The CBOR WG brainstormed this protocol on January 20, 2021.

### **Contributors**

Carsten Bormann  
Universität Bremen TZI  
Germany

Email: [cabo@tzi.org](mailto: cabo@tzi.org)

Josef 'Jeff' Sipek

Email: [jeffpc@josefsipek.net](mailto: jeffpc@josefsipek.net)

### **Author's Address**

Michael Richardson  
Sandelman Software Works

Email: [mcr+ietf@sandelman.ca](mailto: mcr+ietf@sandelman.ca)