CBOR Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 November 2022

M. Richardson
Sandelman Software Works
C. Bormann
Universität Bremen TZI

5 May 2022

On storing CBOR encoded items on stable storage
draft-ietf-cbor-file-magic-12

Abstract

This document defines a stored ("file") format for CBOR data items that is friendly to common file type 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

This Internet-Draft will expire on 6 November 2022.

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. In Unix, everything is a stream of bytes; identifying the contents of a stream of bytes became a heuristic activity.

For instance, the Unix file(1) command, which has existed since 1973 [file], has for decades been able to identify many file formats 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, in classical MacOS, a resource fork was maintained separately from the file data that included file type information; this way, the OS ideally never needed to know anything about the file data contents to determine the media type.

Many other systems do this by file extensions. Many common web servers derive the media-type information from file extensions.

Having a media type associated with the file contents can avoid some of the brittleness of this approach. When files become disconnected from their type information, such as when attempting to do forensics on a damaged system, then being able to identify the type of information that is stored in file can become very important.

A common way to identify the type of a file from its contents is to place a "magic number" at the start of the file contents [MAGIC]. It is noted that in the media type registration template [RFC6838], a magic number is asked for, if available, as is a file extension.

A challenge for the file(1) command is often that it can be confused by the encoding vs. the content. For instance, an Android "apk" (as 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, and may also be identified as a ZIP file.

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. As a CBOR format may use a single CBOR data item or a CBOR sequence of data items [RFC8742], two possible methods of enveloping data are presented; a CBOR Protocol designer will specify one. (A CBOR Protocol is a specification which uses CBOR as its encoding.)

This document also gives advice to designers of CBOR Protocols on choosing one of these mechanisms for identifying their contents. This advice is informative.

A third method is also proposed by which this CBOR format prepended tag is used to identify non-CBOR files. This third method has been placed in Appendix D because it is not about identifying media types containing CBOR-encoded data items. This includes a simple way to derive a magic number to content-formats as defined by [RFC7252], even if not in CBOR form.

Examples of CBOR Protocols currently under development include Concise Software Identification Tags (CoSWID, [I-D.ietf-sacm-coswid]) and Entity Attestation Tokens (EAT, [I-D.ietf-rats-eat]). COSE itself [RFC8152] is considered infrastructure. The encoding of public keys in CBOR as described in [I-D.ietf-cose-cbor-encoded-cert] as _C509_ would benefit from being an identified CBOR Protocol.

A major inspiration for this document is observing the disarray in certain ASN.1 based systems where most files are PEM encoded; these are then all identified by the extension "pem", confusing public keys, private keys, certificate requests, and S/MIME content.

While the envelopes defined in this specification 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. However, 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 can be used to signal that a CBOR Protocol is being used as opposed to a legacy scheme.

1.1. Terminology

Byte is a synonym for octet. The term "byte string" refers to the data item defined in [STD94].

The term "file" is understood to stand in a general way for a stored representation that is somewhat detached from the original context of usage of that representation; its usage in this document encompasses similar units of storage that may have different identification schemes such as partitions or media blocks.

The term "diagnostic notation" refers to the human-readable notation for CBOR data items defined in Section 8 of [STD94] and Appendix G of [RFC8610].

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

The function TN(ct) is defined in Appendix B.

1.2. Requirements for a Magic Number

A magic number is ideally a fingerprint that is unique to a specific 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

This Section presents two enveloping methods. Both use CBOR Tags in a way that results in a deterministic first 8 to 12 bytes. Which one is to be used is up to the CBOR Protocol designer to determine; see Appendix A for some guidance.

2.1. The CBOR Protocol Specific Tag

In both enveloping methods, CBOR Protocol designers need to obtain a CBOR tag for each kind of object that they might store in files. 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 a simple interaction (e.g., via web or email) with IANA, having filled in the small template provided in Section 9.2 of [STD94]. In the template, it is suggested to include a reference to this specification (RFC XXXX) alongside the Description of semantics.
// (Note to RFC Editor: Please replace all occurrences of "RFC XXXX"
// with the RFC number of the present specification and remove this
// note.)

Allocation of the CBOR tag needs to be initiated by the designer of the CBOR Protocol, who can provide a proposed tag number. In order to be in the four-byte range, and so that there are no leading zero bytes in the four-byte encoding of the tag number, the value needs to be in the range 0x01000000 (decimal 16777216) to 0xFFFFFFFF (decimal 4294967295) inclusive. It is further suggested to avoid values that have an embedded zero byte in the four bytes of their binary representation (such as 0x12003456), as these may confuse implementations that treat the magic number as a C string.

The use of a sequence of four US-ASCII [RFC20] codes which are mnemonic to the protocol is encouraged, but not required (there may be reasons to encode other information into the tag; see Appendix B for an example). For instance, Appendix C uses "OPSN" which translates to the tag number 1330664270 registered for it.

For CBOR data items that form 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 pro-actively been allocated in Section 4.3 (see Appendix B 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. This data item is enveloped into two nested tags:

The outer tag is a Self-described CBOR tag, 55799, as described in Section 3.4.6 of [STD94].

The tag content of the outer tag is a second CBOR tag whose tag number has been allocated to describe the specific Protocol involved, as discussed in Section 2.1. The tag content of this inner tag is the single CBOR data item.

This method wraps the CBOR data item as CBOR tags usually do. Applications that need to send the stored CBOR data item across a constrained network may wish to remove the two tags if the type is understood from the protocol context, e.g., from a CoAP Content-Format Option (Section 5.10.3 of [RFC7252]). A CBOR Protocol specification may therefore pick the specific cases where the CBOR Tag Wrapped enveloping method is to be used. For instance, it might specify its use for storing the representation in a local file or for Web access, but not within protocol messages that already provide the necessary context.

2.2.1. Example

To construct an example without registering a new tag, we use the Content-Format number registered in [RFC8428] for application/senml+cbor (as per Registry Content-Formats of [IANA.core-parameters]), the number 112.

Using the technique described in Appendix B, this translates into the tag TN(112) = 1668546929.

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

```
55799(1668546929([{0: "current", 6: 3, 2: 1.5}]))
```

Or in hex:

```
d9 d9f7                       # tag(55799)
   da 63740171                # tag(1668546929)
      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)
```

At the representation level, the unique fingerprint for application/senml+cbor is composed of the 8 bytes d9d9f7da63740171 hex, after which the unadorned CBOR data (81... for the SenML data) is appended.

2.3. Enveloping Method: Labeled CBOR Sequence

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

This method prepends a newly constructed, separate data item to the CBOR Sequence, the _label_.

The label is a nesting of two tags, similar to but distinct from the CBOR Tag Wrapped methods, with an inner tag content of a constant byte string. The total length of the label is 12 bytes.

1. The outer tag is the self-described CBOR Sequence tag, 55800.

2. The inner tag is a CBOR tag, from the First Come First Served space, that uniquely identifies the CBOR Protocol. As with CBOR Tag Wrapped, the use of a four-byte tag is encouraged that encodes without zero bytes.

3. The tag content is a three byte CBOR byte string containing 0x42_4f_52 ('BOR' in diagnostic notation).

The outer tag in the label identifies the file as being a CBOR Sequence, and does so with all the desirable properties explained in Section 3.4.6 of [STD94]. Specifically, it does not appear to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The inner tag in the label identifies which CBOR Protocol is used, as described above.

The inner tag content is a constant byte string which is represented as 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three-byte string 0x42_4f_52 ('BOR' in diagnostic notation).

The actual CBOR Protocol data then follow 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 inner nested tag needs to tag something.

2.3.1. Example

To construct an example without registering a new tag, we use the Content-Format number registered in [RFC9177] for application/missing-blocks+cbor-seq (as per Registry Content-Formats of [IANA.core-parameters]), the number 272.

Using the technique described in Appendix B, this translates into the tag TN(272) = 1668547090.

This is a somewhat contrived example, as this is not a media type that is likely to be committed to storage. Nonetheless, with this tag, missing blocks list 0, 8, 15 would be enveloped as (in diagnostic notation):

```
55800(1668547090('BOR')),
0,
8,
15
```

Or in hex:

```
# CBOR sequence with 4 elements
d9 d9f8         # tag(55800)
   da 63740212  # tag(1668547090)
      43        # bytes(3)
         424f52 # "BOR"
00 # unsigned(0)
08 # unsigned(8)
0f # unsigned(15)
```

At the representation level, the unique fingerprint for application/missing-blocks+cbor-seq is composed of the 8 bytes d9d9f8da63740212 hex, after which the unadorned CBOR sequence (00... for the missing block list given) is appended.

3. Security Considerations

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

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

Protective applications (that check data) cannot rely on the applications they try to protect (that use the data) to make exactly the same decisions in recognizing file formats. (This is an instance of a check vs. use issue.) For example, end-point assessment technologies should not solely rely on the labeling approaches described in this document to decide whether to inspect a given file. Similarly, depending on operating systems configurations and related properties of the execution environment the labeling might influence the default application used to process a file in a way that may not be predicted by a protective application.

4. IANA Considerations

These IANA considerations are entirely about CBOR Tags, in the registry CBOR Tags of [IANA.cbor-tags].

Section 4.1 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 C). Section 4.3 allocates a CBOR tag for each actual or potential CoAP Content-Format number (examples are in Appendix B).

4.1. Labeled CBOR Sequence Tag

IANA has allocated tag 55800 as the tag for the Labeled CBOR Sequence Enveloping Method from the CBOR Tags Registry. IANA is asked to update this tag registration to point to this document.

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 [STD94]).

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 [STD94] 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

4.2. CBOR-Labeled Non-CBOR Data Tag

IANA is requested to allocate tag 55801 as the tag for the CBOR-Labeled Non-CBOR Data Enveloping Method (Appendix D) from the CBOR Tags Registry. IANA is asked to update this tag registration to point to this document.

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 [STD94]).

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 [STD94] 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-Labeled Non-CBOR Data label.

4.3. CBOR Tags for CoAP Content-Format Numbers

IANA is requested to allocate the tag numbers 1668546817 (0x63740101) to 1668612095 (0x6374ffff) as follows:

Data Item:
byte string or any CBOR data item (see Appendix B of RFC XXXX)

Semantics:
the representation of content-format ct < 65025 is indicated by tag number
TN(ct) = 0x63470101 + (ct / 255) * 256 + ct % 255

Reference:
RFC XXXX

The Registry for Content-Formats of [IANA.core-parameters] has been defined in Section 12.3 of [RFC7252].

5. References

5.1. Normative References

[C] International Organization for Standardization, "Information technology Programming languages C", ISO/IEC 9899:2018, Fourth Edition, June 2018, <https://www.iso.org/standard/74528.html>.

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

[STD94] 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>.

5.2. Informative References

[file] Wikipedia, "file (command)", 20 January 2021, <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. Furuhed, "CBOR Encoded X.509 Certificates (C509 Certificates)", Work in Progress, Internet-Draft, draft-ietf-cose-cbor-encoded-cert-03, 10 January 2022, <https://www.ietf.org/archive/id/draft-ietf-cose-cbor-encoded-cert-03.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-12, 24 February 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-12.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-21, 7 March 2022, <https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-21.txt>.

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

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

[MAGIC] Ritchie, D., "archive (library) file format", in Bell Labs, Unix Programmer’s Manual, First Edition: File Formats, 3 November 1971, <https://www.bell-labs.com/usr/dmr/www/man51.pdf#page=4>.

[RFC20] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[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>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[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>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

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

[RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018, <https://www.rfc-editor.org/info/rfc8428>.

[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>.

[RFC9177] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", RFC 9177, DOI 10.17487/RFC9177, March 2022, <https://www.rfc-editor.org/info/rfc9177>.

[X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, February 2021.

Appendix A. Advice to Protocol Designer

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 ([I-D.ietf-cose-cbor-encoded-cert]) or CBOR format, a recorded sensor reading stored for later transmission, or a COVID-19 vaccination certificate that needs to be displayed in QR code form.

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

The Labeled CBOR 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 Labeled CBOR 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. The three byte entry is not of the format the program would normally have processed, so it may be a surprise. On the other hand, CBOR parsers are generally tolerant of tags that appear: many of them will process extra tags, making unknown tags available as meta information. A program that is not expecting those tags may just ignore those extra tags.

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 [RFC8017] 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 ([X.690], 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 stored 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 files or similar units of storage, 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 B for media-types with CoAP Content-Format numbers.

Here are some considerations in the form of survey questions:

A.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 stored 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 Labeled CBOR Sequence format should be adopted for the stored format.

A.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 Labeled CBOR Sequence format may be easily retrofitted.

The program involved may throw errors or warnings on the Labeled CBOR Sequence if they have not yet been updated, but this may not be a problem.

There are situations where multiple objects may be concatenated into a single file. If each object is preceded by a Labeled CBOR Sequence label then there may be multiple such labels in the file.

A protocol based on CBOR Sequences may specify that Labeled CBOR Sequence labels can occur within a CBOR Sequence, possibly even to switch to data items following in the sequence that are of a different type.

If the CBOR Sequence based protocol does not define the semantics for or at least tolerate embedded labels, care must be taken when concatenating Labeled CBOR Sequences to remove the label from all but the first part.

> As an example from legacy PEM encoded PKIX certificates, many programs accept a series of PKIX certificates in a single file in order to set up a certificate chain. The file would contain not just the End-Entity (EE) certificate, but also any subordinate certification authorities (CA) needed to validate the EE. This mechanism actually only works for PEM encoded certificates, and not DER encoded certificates. One of the reasons for this specification is to make sure that CBOR encoded certificates do not suffer from this problem.
>
> As an example of mixing of types, some TLS server programs also can accept both their PEM encoded private key, and their PEM encoded certificate in the same file.

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

A.3. Are there tags at the start?

If the Protocol expects to use other tags at its top-level, then the use of the CBOR Tag Wrapped format may be easy to explain at the same place in the protocol description.

Appendix B. CBOR Tags for CoAP Content Formats

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.

Outside of a transfer protocol that indicates the Content-Format for a representation, it may be necessary to identify the Content-Format of the representation when it is stored in a file, in firmware, or when debugging.

This specification allocates CBOR tag numbers 1668546817 (0x63740101) to 1668612095 (0x6374FFFF) for the tagging of representations of specific content formats.

Using tags from this range, a byte string that is to be interpreted as a representation of Content-Format number ct, with ct < 65025 (255*255), can be identified by enclosing it in a tag with tag number TN(ct) where:

```
TN(ct) = 0x63470101 + (ct / 255) * 256 + ct % 255.
```

(where +, *, / and % stand for integer addition, multiplication, division and remainder as in the programming language C [C].)

> This formula avoids the use of zero bytes in the representation of the tag number.
>
> Note that no tag numbers are assigned for Content-Format numbers in the range 65025 ct 65535. (This range is in the range reserved by Section 12.3 of [RFC7252] for experimental use. The overlap of 25 code points between this experimental range with the range this appendix defines tag numbers for can be used for experiments that want to employ a tag number.)

Exceptionally, when used immediately as tag content of one of the tags 55799, 55800, or 55801, the tag content is as follows:

Tag 55799 (Section 2.2): One of:

1. The CBOR data item within the representation (without byte string wrapping). This only works for Content Formats that are represented by a single CBOR data item in identity content-coding.

2. The data items in the CBOR sequence within the representation, without byte string wrapping, but wrapped in a CBOR array. This works for Content Formats that are represented by a CBOR sequence in identity content-coding.

Tags 55800 (Section 2.3) or 55801 (Appendix D): the byte string 'BOR', signifying that the representation of the given content-format follows in the file, in the way defined for these tags.

B.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 TN(112) = 1668546929.

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

  ```
  senml-cbor = #6.1668546929(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 [RFC9177].

  The corresponding tag number is TN(272) = 1668547090.

  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 = #6.1668547090(bstr)
  ```

Appendix C. 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 D. Using CBOR Labels for non-CBOR data

The CBOR-Labeled non-CBOR data 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 B).

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

1. The outer tag is the CBOR-Labeled Non-CBOR Data tag, 55801.

2. The inner tag is a CBOR tag, from the First Come First Served space, that uniquely identifies the CBOR Protocol. As with CBOR Tag Wrapped, the use of a four-byte tag is encouraged that encodes without zero bytes.

3. The tag content is a three byte CBOR byte string containing 0x42_4F_52 ('BOR' in diagnostic notation).

The outer tag in the label identifies the file as being file as being prefixed by a non-CBOR data label, and does so with all the desirable properties explained in Section 3.4.6 of [STD94]. Specifically, it does not appear to conflict with any known file types, and it is not valid Unicode in any Unicode encoding.

The inner tag in the label identifies which non-CBOR Protocol is used.

The inner tag content is a constant byte string which is represented as 0x43_42_4f_52, the ASCII characters "CBOR", which is the CBOR encoded data item for the three-byte string 0x42_4f_52 ('BOR' in diagnostic notation).

The actual non-CBOR Protocol data then follow directly appended to the CBOR representation of the header. This allows the application to trivially remove the header item with the two nested tags and the byte string.

As with the Labeled CBOR Sequence {#sequences}, this choice of the tag content places the ASCII characters "CBOR" prominently into the header.

D.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 TN(432) = 1668547250.

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

  ```
  td-json-header = #6.55801(#6.1668547250('BOR'))
  ```

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

  The corresponding tag number is TN(11050) = 1668557910.

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

  ```
  json-deflate-header = #6.55801(#6.1668557910('BOR'))
  ```

Appendix E. 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