Workgroup: Network Working Group
Internet-Draft: draft-ietf-cbor-packed-03
Published: 13 August 2021
Intended Status: Informational
Expires: 14 February 2022
Authors: C. Bormann
Universität Bremen TZI

Packed CBOR

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

CBOR does not provide any forms of data compression. CBOR data items, in particular when generated from legacy data models often allow considerable gains in compactness when applying data compression. While traditional data compression techniques such as DEFLATE (RFC 1951) can work well for CBOR encoded data items, their disadvantage is that the receiver needs to unpack the compressed form to make use of data.

This specification describes Packed CBOR, a simple transformation of a CBOR data item into another CBOR data item that is almost as easy to consume as the original CBOR data item. A separate decompression step is therefore often not required at the receiver.

Note to Readers

This is a working-group draft of the CBOR working group of the IETF, https://datatracker.ietf.org/wg/cbor/about/. Discussion takes places on the github repository https://github.com/cbor-wg/cbor-packed and on the CBOR WG mailing list, https://www.ietf.org/mailman/listinfo/cbor.

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 14 February 2022.

Table of Contents

1. Introduction

(TO DO, expand on text from abstract here; move references here and neuter them in the abstract as per Section 4.3 of [RFC7322].)

The specification defines a transformation from a Packed CBOR data item to the original CBOR data item; it does not define an algorithm for an actual packer. Different packers can differ in the amount of effort they invest in arriving at a minimal packed form.

Packed CBOR can employ two kinds of optimization:

- item sharing: substructures (data items) that occur repeatedly in the original CBOR data item can be collapsed to a simple reference to a common representation of that data item. The processing required during consumption is limited to following that reference.

- affix sharing: data items (strings, containers) that share a prefix or suffix (affix) can be replaced by a reference to a common affix plus the rest of the data item. For strings, the processing required during consumption is similar to following the affix reference plus that for an indefinite-length string.

A specific application protocol that employs Packed CBOR might allow both kinds of optimization or limit the representation to item sharing only.

Packed CBOR is defined in two parts: Referencing packing tables (Section 2) and setting up packing tables (Section 3).

1.1. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL** NOT", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Packed reference: A shared item reference or an affix reference

Shared item reference: A reference to a shared item as defined in Section 2.2

Affix reference: A reference that combines an affix item as defined in Section 2.3.

Affix: Prefix or suffix.

Packing tables: The triple of a shared item table, a prefix table, and a suffix table.

Expansion: The result of applying a packed reference in the context of given Packing tables.

The definitions of [RFC8949] apply. The term "byte" is used in its now customary sense as a synonym for "octet". Where bit arithmetic is explained, this document uses the notation familiar from the programming language C (including C++14's 0bnnn binary literals), except that, in the plain text form of this document, the operator

"^" stands for exponentiation, and, in the HTML and PDF versions, subtraction and negation are rendered as a hyphen ("-", as are various dashes).

2. Packed CBOR

This section describes the packing tables, their structure, and how they are referenced.

2.1. Packing Tables

At any point within a data item making use of Packed CBOR, there is a Current Set of packing tables that applies.

There are three packing tables in a Current Set:

* Shared item table

* Prefix table

* Suffix table

Without any table setup, all these tables are empty arrays. Table setup can cause these arrays to be non-empty, where the elements are (potentially themselves packed) data items. Each of the tables is indexed by an unsigned integer (starting from 0), which may be computed from information in tags and their content as well as from CBOR simple values.

2.2. Referencing Shared Items

Shared items are stored in the shared item table of the Current Set.

The shared data items are referenced by using the reference data items in Table 1. When reconstructing the original data item, such a reference is replaced by the referenced data item, which is then recursively unpacked.

reference	table index
Simple value 0-15	0-15
Tag 6(unsigned integer N)	16 + 2*N
Tag 6(negative integer N)	16 - 2*N - 1

Table 1: Referencing Shared Values

As examples in CBOR diagnostic notation (Section 8 of [RFC8949]), the first 22 elements of the shared item table are referenced by simple(0), simple(1), ... simple(15), 6(0), 6(-1), 6(1), 6(-2), 6(2), 6(-3). (The alternation between unsigned and negative integers

for even/odd table index values makes systematic use of shorter integer encodings first.)

Taking into account the encoding of these referring data items, there are 16 one-byte references, 48 two-byte references, 512 three-byte references, 131072 four-byte references, etc. As integers can grow to very large (or negative) values, there is no practical limit to how many shared items might be used in a Packed CBOR item.

Note that the semantics of Tag 6 depend on its content: An integer turns the tag into a shared item reference, a string or container (map or array) into a prefix reference (see Table 2).

2.3. Referencing Affix Items

Prefix items are stored in the prefix table of the Current Set; suffix items are stored in the suffix table of the Current Set. We collectively call these items affix items; when referencing, which of the tables is actually used depends on whether a prefix or a suffix reference was used.

prefix reference	table index
Tag 6(suffix)	0
Tag 225-255(suffix)	1-31
Tag 28704-32767(suffix)	32-4095
Tag 1879052288-2147483647(suffix)	4096-268435455

Table 2: Referencing Prefix Values

suffix reference	table index
Tag 216-223(prefix)	0-7
Tag 27647-28671(prefix)	8-1023
Tag 1811940352-1879048191(prefix)	1024-67108863

Table 3: Referencing Suffix Values

Affix data items are referenced by using the data items in Table 2 and Table 3. Each of these implies the table used (prefix or suffix), a table index (an unsigned integer) and contains a "rump item". When reconstructing the original data item, such a reference is replaced by a data item constructed from the referenced affix data item (affix, which might need to be recursively unpacked first) "concatenated" with the tag content (rump, again possibly recursively unpacked).

*For a rump of type array and map, the affix also needs to be an array or a map. For an array, the elements from the prefix are prepended, and the elements from a suffix are appended to the rump array. For a map, the entries in the affix are added to those of the rump; prefix and suffix references differ in how

entries with identical keys are combined: for prefix references, an entry in the rump with the same key as an entry in the affix overrides the one in the affix, while for suffix references, an entry in the affix overrides an entry in the rump that has the same key.

ISSUE: Not sure that we want to use the efficiencies of overriding, but having default values supplied out of a dictionary to be overridden by a rump sounds rather handy. Note that there is no way to remove a map entry from the table.

*For a rump of one of the string types, the affix also needs to be one of the string types; the bytes of the strings are concatenated as specified (prefix + rump, rump + suffix). The result of the concatenation gets the type of the rump; this way a single affix can be used to build both byte and text strings, depending on what type of rump is being used.

As a contrived (but short) example, if the prefix table is ["foobar", "foob", "fo"], the following prefix references will all unpack to "foobart": 6("t"), 224("art"), 225("obart") (the last example is not an optimization).

Taking into account the encoding, there is one single-byte prefix reference, 31 (2^5-2^0) two-byte references, 4064 (2^{12}-2^5) three-byte references, and 26843160 (2^{28}-2^{12}) five-byte references for prefixes. 268435455 (2^{28}) is an artificial limit, but should be high enough that there, again, is no practical limit to how many prefix items might be used in a Packed CBOR item. The numbers for suffix references are one quarter of those, except that there is no single-byte reference and 8 two-byte references.

Rationale: Experience suggests that prefix packing might be more likely than suffix packing. Also for this reason, there is no intent to spend a 1+0 tag value for suffix matching.

2.4. Discussion

This specification uses up a large number of Simple Values and Tags, in particular one of the rare one-byte tags and half of the one-byte simple values. Since the objective is compression, this is warranted if and only if there is consensus that this specific format could be useful for a wide area of applications, while maintaining reasonable simplicity in particular at the side of the consumer.

A maliciously crafted Packed CBOR data item might contain a reference loop. A consumer/decompressor **MUST** protect against that.

Different strategies for decoding/consuming Packed CBOR are available.
For example:

- the decoder can decode and unpack the packed item, presenting an unpacked data item to the application. In this case, the onus of dealing with loops is on the decoder. (This strategy generally has the highest memory consumption, but also the simplest interface to the application.) Besides avoiding getting stuck in a reference loop, the decoder will need to control its resource allocation, as data items can "blow up" during unpacking.

- the decoder can be oblivious of Packed CBOR. In this case, the onus of dealing with loops is on the application, as is the entire onus of dealing with Packed CBOR.

- hybrid models are possible, for instance: The decoder builds a data item tree directly from the Packed CBOR as if it were oblivious, but also provides accessors that hide (resolve) the packing. In this specific case, the onus of dealing with loops is on the accessors.

In general, loop detection can be handled in a similar way in which loops of symbolic links are handled in a file system: A system wide limit (often 31 or 40 indirections for symbolic links) is applied to any reference chase.
ISSUE: The present specification does nothing to help with the packing of CBOR sequences [RFC8742]; maybe it should.

3. Table Setup

The packing references described in Section 2 assume that packing tables have been set up.

By default, all three tables are empty (zero-length arrays).

Table setup can happen in one of two ways:

- By the application environment, e.g., a media type. These can define tables that amount to a static dictionary that can be used in a CBOR data item for this application environment. Note that, without this information, a data item that uses such a static dictionary can be decoded at the CBOR level, but not fully unpacked. The table setup mechanisms provided by this document are defined in such a way that an unpacker can at least recognize if this is the case.

- By one or more tags enclosing the packed content. These can be defined to add to the packing tables that already apply to the

tag. Usually, the semantics of the tag will be to prepend items to one of the tables. Note that it may be useful to leave a particular efficiency tier alone and only prepend to a higher tier; e.g., a tag could insert shared items at table index 16 and shift anything that was already there further down in the array while leaving index 0 to 15 alone. Explicit additions by tag can combine with application-environment supplied tables that apply to the entire CBOR data item.

For table setup, the present specification only defines a single tag, which operates by prepending to the (by default empty) tables.

We could also define a tag for dictionary referencing (or include that in the basic packed CBOR), but the desirable details are likely to vary considerably between applications. A URI-based reference would be easy to add, but might be too inefficient when used in the likely combination with an ni: URI [RFC6920].

3.1. Basic Packed CBOR

A predefined tag for packing table setup is defined in CDDL [RFC8610] as in Figure 1:

```
Basic-Packed-CBOR = #6.51([[*shared-item], [*prefix-item],
                           [*suffix-item], rump])
rump = any
prefix-item = any
suffix-item = any
shared-item = any
```

Figure 1: Packed CBOR in CDDL

(This assumes the allocation of tag number 51 for this tag.)

The arrays given as the first, second, and third element of the content of the tag 51 are prepended to the tables for shared items, prefixes, and suffixes that apply to the entire tag (by default empty tables).

The original CBOR data item can be reconstructed by recursively replacing shared, prefix, and suffix references encountered in the rump by their expansions.

Packed item references in the newly constructed (low-numbered) parts of the table need to be interpreted in the number space of that table (which includes the, now higher-numbered inherited parts), while references in any existing, inherited (higher-numbered) part continue to use the (more limited) number space of the inherited table.

4. IANA Considerations

In the registry "CBOR Tags" [IANA.cbor-tags], IANA is requested to allocate the tags defined in Table 4.

Tag	Data Item	Semantics	Reference
6	integer (for shared); text string, byte string, array, map (for prefix)	Packed CBOR: shared/ prefix	draft-ietf-cbor-packed
225-255	text string, byte string, array, map	Packed CBOR: prefix	draft-ietf-cbor-packed
28704-32767	text string, byte string, array, map	Packed CBOR: prefix	draft-ietf-cbor-packed
1879052288-2147483647	text string, byte string, array, map	Packed CBOR: prefix	draft-ietf-cbor-packed
216-223	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed
27647-28671	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed
1811940352-1879048191	text string, byte string, array, map	Packed CBOR: suffix	draft-ietf-cbor-packed

Table 4: Values for Tag Numbers

In the registry "CBOR Simple Values" [IANA.cbor-simple-values], IANA is requested to allocate the simple values defined in Table 5.

Value	Semantics	Reference
0-15	Packed CBOR: shared	draft-ietf-cbor-packed

Table 5: Simple Values

5. Security Considerations

The security considerations of [RFC8949] apply.

Loops in the Packed CBOR can be used as a denial of service attack, see Section 2.4.

As the unpacking is deterministic, packed forms can be used as signing inputs. (Note that if external dictionaries are added to cbor-packed, this requires additional consideration.)

6. References

6.1. Normative References

[IANA.cbor-simple-values] IANA, "Concise Binary Object Representation (CBOR) Simple Values", <http://www.iana.org/assignments/cbor-simple-values>.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

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

[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

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <https://www.rfc-editor.org/info/rfc7322>.

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

Appendix A. Examples

The (JSON-compatible) CBOR data structure depicted in Figure 2, 400 bytes of binary CBOR, could lead to a packed CBOR data item depicted in Figure 3, ~309 bytes. Note that this particular example does not lend itself to prefix compression.

```
{ "store": {
    "book": [
      { "category": "reference",
        "author": "Nigel Rees",
        "title": "Sayings of the Century",
        "price": 8.95
      },
      { "category": "fiction",
        "author": "Evelyn Waugh",
        "title": "Sword of Honour",
        "price": 12.99
      },
      { "category": "fiction",
        "author": "Herman Melville",
        "title": "Moby Dick",
        "isbn": "0-553-21311-3",
        "price": 8.99
      },
      { "category": "fiction",
        "author": "J. R. R. Tolkien",
        "title": "The Lord of the Rings",
        "isbn": "0-395-19395-8",
        "price": 22.99
      }
    ],
    "bicycle": {
      "color": "red",
      "price": 19.95
    }
  }
}
```

Figure 2: Example original CBOR data item

```
51(["price", "category", "author", "title", "fiction", 8.95, "isbn"],
   /  0          1         2         3         4       5      6   /
   [], [],
   [{"store": {
      "book": [
        {simple(1): "reference", simple(2): "Nigel Rees",
         simple(3): "Sayings of the Century", simple(0): simple(5)},
        {simple(1): simple(4), simple(2): "Evelyn Waugh",
         simple(3): "Sword of Honour", simple(0): 12.99},
        {simple(1): simple(4), simple(2): "Herman Melville",
         simple(3): "Moby Dick", simple(6): "0-553-21311-3",
         simple(0): simple(5)},
        {simple(1): simple(4), simple(2): "J. R. R. Tolkien",
         simple(3): "The Lord of the Rings",
         simple(6): "0-395-19395-8", simple(0): 22.99}],
      "bicycle": {"color": "red", simple(0): 19.95}}}])
```

Figure 3: Example packed CBOR data item

The (JSON-compatible) CBOR data structure below has been packed with shared item and (partial) prefix compression only.

```
{
  "name": "MyLED",
  "interactions": [
    {
      "links": [
        {
          "href":
           "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueRed",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueRed",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "href":
           "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueGreen",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "rgbValueGreen",
      "writable": true,
      "@type": [
        "Property"
      ]
    },
    {
      "links": [
        {
          "href":
           "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueBlue",
          "mediaType": "application/json"
        }
      ],
```

```
  "outputData": {
    "valueType": {
      "type": "number"
    }
  },
  "name": "rgbValueBlue",
  "writable": true,
  "@type": [
    "Property"
  ]
},
{
  "links": [
    {
      "href":
       "http://192.168.1.103:8445/wot/thing/MyLED/rgbValueWhite",
      "mediaType": "application/json"
    }
  ],
  "outputData": {
    "valueType": {
      "type": "number"
    }
  },
  "name": "rgbValueWhite",
  "writable": true,
  "@type": [
    "Property"
  ]
},
{
  "links": [
    {
      "href":
       "http://192.168.1.103:8445/wot/thing/MyLED/ledOnOff",
      "mediaType": "application/json"
    }
  ],
  "outputData": {
    "valueType": {
      "type": "boolean"
    }
  },
  "name": "ledOnOff",
  "writable": true,
  "@type": [
    "Property"
  ]
},
```

```
    {
      "links": [
        {
          "href":
"http://192.168.1.103:8445/wot/thing/MyLED/colorTemperatureChanged",
          "mediaType": "application/json"
        }
      ],
      "outputData": {
        "valueType": {
          "type": "number"
        }
      },
      "name": "colorTemperatureChanged",
      "@type": [
        "Event"
      ]
    }
  ],
  "@type": "Lamp",
  "id": "0",
  "base": "http://192.168.1.103:8445/wot/thing",
  "@context":
   "http://192.168.1.102:8444/wot/w3c-wot-td-context.jsonld"
}
```

Figure 4: Example original CBOR data item

```
51([/shared/["name", "@type", "links", "href", "mediaType",
           /  0       1       2        3         4 /
    "application/json", "outputData", {"valueType": {"type":
         /  5               6               7 /
    "number"}}, ["Property"], "writable", "valueType", "type"],
              /   8            9           10           11 /
   /prefix/ ["http://192.168.1.10", 6("3:8445/wot/thing"),
             / 6                        225 /
   225("/MyLED/"), 226("rgbValue"), "rgbValue",
     / 226             227           228     /
   {simple(6): simple(7), simple(9): true, simple(1): simple(8)}],
     / 229 /
   /suffix/ [],
   /rump/ {simple(0): "MyLED",
           "interactions": [
   229({simple(2): [{simple(3): 227("Red"), simple(4): simple(5)}],
    simple(0): 228("Red")}),
   229({simple(2): [{simple(3): 227("Green"), simple(4): simple(5)}],
    simple(0): 228("Green")}),
   229({simple(2): [{simple(3): 227("Blue"), simple(4): simple(5)}],
    simple(0): 228("Blue")}),
   229({simple(2): [{simple(3): 227("White"), simple(4): simple(5)}],
    simple(0): "rgbValueWhite"}),
   {simple(2): [{simple(3): 226("ledOnOff"), simple(4): simple(5)}],
    simple(6): {simple(10): {simple(11): "boolean"}}, simple(0):
    "ledOnOff", simple(9): true, simple(1): simple(8)},
   {simple(2): [{simple(3): 226("colorTemperatureChanged"),
    simple(4): simple(5)}], simple(6): simple(7), simple(0):
    "colorTemperatureChanged", simple(1): ["Event"]}],
     simple(1): "Lamp", "id": "0", "base": 225(""),
     "@context": 6("2:8444/wot/w3c-wot-td-context.jsonld")}])
```

Figure 5: Example packed CBOR data item

Acknowledgements

CBOR packing was originally invented with the rest of CBOR, but did not make it into [RFC7049], the predecessor of [RFC8949]. Various attempts to come up with a specification over the years didn't proceed. In 2017, Sebastian Käbisch proposed investigating compact representations of W3C Thing Descriptions, which prompted the author to come up with essentially the present design.

Author's Address

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen

Germany

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