Workgroup: Network Working Group
Internet-Draft: draft-ietf-cbor-packed-07
Published: 24 July 2022
Intended Status: Standards Track
Expires: 25 January 2023
Authors:
C. Bormann
Universität Bremen TZI

Packed CBOR

Abstract

The Concise Binary Object Representation (CBOR, RFC 8949 == STD 94) 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 decompress the compressed form to make use of the 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.

The present version (-07) adds the concept of Tag Equivalence as initially discussed at the CBOR Interim meeting 12 in 2022 and further in PR #6, for discussion before and at IETF 114.

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-packed/.

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/. Subscribe at https://www.ietf.org/mailman/listinfo/cbor/.

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

Status of This Memo

This Internet-Draft will expire on 25 January 2023.

Copyright Notice

Table of Contents

Author's Address

1. Introduction

The Concise Binary Object Representation (CBOR, [STD94]) 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 [RFC1951] can work well for CBOR encoded data items, their disadvantage is that the receiver needs to decompress the compressed form to make use of the 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.

This document defines the Packed CBOR format by specifying the transformation from a Packed CBOR data item to the original CBOR data item; it does not define an algorithm for a packer. Different packers can differ in the amount of effort they invest in arriving at a minimal packed form; often, they simply employ the sharing that is natural for a specific application.

Packed CBOR can make use of 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.

* argument sharing: application of a function with two arguments, one of which is shared. Data items (strings, containers) that share a prefix or suffix (affix), or more generally data items that can be constructed from a function taking a shared argument and a rump data item, can be replaced by a reference to the shared argument plus a rump data item. For strings and the default "concatenation" function, the processing required during consumption is similar to following the argument 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 argument reference.

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

Argument reference: A reference that combines a shared argument with a rump item as defined in Section 2.3.

Affix: Prefix or suffix, used as an argument in an argument reference employing the default function "concatenation".

Function reference: An argument reference that uses a tag for argument, rump, or both, causing the application of a function to reconstruct the data item.

Packing tables: The pair of a shared item table and an argument table.

Current set: The packing tables in effect at the data item under consideration.

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

The definitions of [STD94] apply. Specifically: The term "byte" is used in its now customary sense as a synonym for "octet"; "byte strings" are CBOR data items carrying a sequence of zero or more (binary) bytes, while "text strings" are CBOR data items carrying a sequence of zero or more Unicode code points (more precisely: Unicode scalar values), encoded in UTF-8 [STD63].

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 two packing tables in a Current Set:

* Shared item table

* Argument table

Without any table setup, these two 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). Such an index may be derived 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 [STD94]), 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 -- "zigzag encoding" -- 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 CBOR integers can grow to very large (or very 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 tag content: An integer turns the tag into a shared item reference, whereas a string or container (map or array) turns it into a straight (prefix) reference (see Table 2). Note also that the tag content of Tag 6 may itself be packed, so it may need to be unpacked to make this determination.

2.3. Referencing Argument Items

The argument table serves as a common table that can be used for argument references, i.e., for concatenation as well as references involving a function tag.

When referencing an argument, a distinction is made between straight and inverted references; if no function tag is involved, a straight reference combines a prefix out of the argument table with the rump data item, and an inverted reference combines a rump data item with a suffix out of the argument table.

straight reference	table index
Tag 6(straight rump)	0
Tag 224-255(straight rump)	0-31
Tag 28704-32767(straight rump)	32-4095
Tag 1879052288-2147483647(straight rump)	4096-268435455

Table 2: Straight Referencing (e.g., Prefix) Arguments

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

Table 3: Inverted Referencing (e.g., Suffix) Arguments

Argument data items are referenced by using the reference data items in Table 2 and Table 3.

The tag number of the reference indicates a table index (an unsigned integer) leading to the "argument"; the tag content of the reference is the "rump item".

When reconstructing the original data item, such a reference is replaced by a data item constructed from the argument data item found in the table (argument, which might need to be recursively unpacked first) and the rump data item (rump, again possibly recursively unpacked).

Separate from the tag used as a reference, a tag ("function tag") may be involved to supply a function to be used in resolving the reference. It is crucial not to confuse reference tag and, if present, function tag.

A straight reference uses the argument as the provisional left hand side and the rump data item as the right hand side. An inverted reference uses the rump data item as the provisional left hand side and the argument as the right hand side.

In both cases, the provisional left hand side is examined. If it is a tag ("function tag"), it is "unwrapped": The function tag's tag number is established as the function to be applied, and the tag content is kept as the unwrapped left hand side. If the provisional left hand side is not a tag, it is kept as the unwrapped left hand side, and the function to be applied is concatenation, as defined below. The right hand side is taken as is as the unwrapped right hand side.

If a function tag was given, the reference is replaced by the result of applying the unpacking function to be computed to the left and right hand sides. The unpacking function is defined by the definition of the tag number supplied. If that definition does not define an unpacking function, the result of the unpacking is not valid.

If no function tag was given, the reference is replaced by the left hand side "concatenated" with the right hand side, where concatenation is defined as in Section 2.4.

As a contrived (but short) example, if the argument table is ["foobar", h'666f6f62', "fo"], each of the following straight (prefix) references will unpack to "foobart": 6("t"), 225("art"), 226("obart") (the byte string h'666f6f62' == 'foob' is concatenated into a text string, and the last example is not an optimization).

Note that table index 0 of the argument table can be referenced both with tag 6 and tag 224, however tag 6 with an integer content is used for shared item references (see Table 1), so to combine index 0 with an integer rump, tag 224 needs to be used.

Taking into account the encoding and ignoring the less optimal tag 224, there is one single-byte straight (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 straight references. 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 inverted (suffix) references are one quarter of those, except that there is no single-byte reference and 8 two-byte references.

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

2.4. Concatenation

The concatenation function is defined as follows:

* If both left hand side and right hand side are arrays, the result of the concatenation is an array with all elements of the left-hand-side array followed by the elements of the right-hand side array.

* If both left hand side and right hand side are maps, the result of the concatenation is a map that is initialized with a copy of the left-hand-side map, and then filled in with the members of the right-hand side map, replacing any existing members that have the same key.

  NOTE: One application of the rule for straight references is to supply default values out of a dictionary, which can then be overridden by the entries in the map supplied as the rump data item. Note that this pattern provides no way to remove a map entry from the prefix table entry.

* If both left hand side and right hand side are one of the string types (not necessarily the same), the bytes of the left hand side are concatenated with the bytes of the right hand side. Byte strings concatenated with text strings need to contain valid UTF-8 data. The result of the concatenation gets the type of the unwrapped rump data item; this way a single argument table entry can be used to build both byte and text strings, depending on what type of rump is being used.

* If one side is one of the string types, and the other side is an array, the result of the concatenation is equivalent to the

application of the "join" function (Section 4.1) to the string as the left hand side and the array as the right hand side. The original right hand side of the concatenation determines the string type of the result.

* Other type combinations of left hand side and right hand side are not valid.

2.5. Discussion

This specification uses up a large number of Simple Values and Tags, in particular one of the rare one-byte tags and two thirds of the one-byte simple values. Since the objective is compression, this is warranted only based on a 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.

NOTE: The present specification does nothing to help with the packing of CBOR sequences [RFC8742]; maybe such a specification should be added.

3. Table Setup

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

By default, both 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. Each tag is usually defined to build an augmented table by adding to the packing tables that already apply to the tag, and to apply the resulting augmented table when unpacking the tag content. Usually, the semantics of the tag will be to prepend items to one or more of the tables. (The specific behavior of any such tag, in the presence of a table applying to it, needs to be carefully specified.)

  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.

  Packed item references in the newly constructed (low-numbered) parts of the table are usually 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.

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 = #6.113([[*shared-item], [*argument-item], rump])
rump = any
argument-item = any
shared-item = any
```

Figure 1: Packed CBOR in CDDL

(This assumes the allocation of tag number 113 ('q') for this tag.)

The arrays given as the first and second element of the content of the tag 113 are prepended to the tables for shared items and arguments that apply to the entire tag (by default empty tables). As discussed in the introduction to this section, references in the supplied new arrays use the new number space (where inherited items are shifted by the new items given), while the inherited items themselves use the inherited number space (so their semantics do not change by the mere action of inheritance).

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

4. Function Tags

Function tags that occur in an argument or a rump supply the semantics for reconstructing a data item from their tag content and the non-dominating rump or argument, respectively. The present specification defines a pair of function tags.

4.1. Join Function Tags

Tag 106 ('j') defines the "join" unpacking function, based on the concatenation function (Section 2.4).

The join function expects an item that can be concatenated as its left hand side, and an array of such items as its right hand side. Joining works by sequentially applying the concatenation function to the elements of the right-hand-side array, interspersing the left hand side as the "joiner".

An example in functional notation: join(", ", ["a", "b", "c"]) returns "a, b, c".

For a right hand side of one or more elements, the first element determines the type of the result when text strings and byte strings are mixed in the argument. For a right hand side of one element, the joiner is not used, and that element returned. For a right hand side of zero elements, a neutral element is generated based on the type of the joiner (empty text/byte string for a text/byte string, empty array for an array, empty map for a map).

For an example, we assume this unpacked data item:

```
["https://packed.example/foo.html",
 "coap:://packed.example/bar.cbor",
 "mailto:support@packed.example"]
```

A packed form of this using straight references could be:

```
113([ [],
  [106("packed.example")],
  [6(["https://", "/foo.html"]),
   6(["coap://", "/bar.cbor"]),
   6(["mailto:support@", ""])]
])
```

Tag 105 ('i') defines the "ijoin" unpacking function, which is exactly like that of tag 106, except that the left hand side and right hand side are interchanged ('i').

A packed form of the first example using inverted references and the ijoin tag could be:

```
113([ [],
  ["packed.example"],
  [216(105(["https://", "/foo.html"]),
   216(105(["coap://", "/bar.cbor"]),
   216("mailto:support@")]
])
```

A packed form of an array with many URIs that reference SenML items from the same place could be:

```
113([ [],
  [105(["coaps://[2001::db8::1]/s/", ".senml"])],
  [6("temp-freezer"),
   6("temp-fridge"),
   6("temp-ambient")]
])
```

5. Tag Validity: Tag Equivalence Principle

In Section 5.3.2 of [STD94], the validity of tags is defined in terms of type and value of their tag content. The CBOR Tag registry [IANA.cbor-tags] Section 9.2 of [STD94] allows recording the "data item" for a registered tag, which is usually an abbreviated description of the top-level data type allowed for the tag content.

In other words, in the registry, the validity of a tag of a given tag number is described in terms of the top-level structure of the data carried in the tag content. The description of a tag might add further constraints for the data item. But in any case, a tag definition can only specify validity based on the structure of its tag content.

In Packed CBOR, a reference tag might be "standing in" for the actual tag content of an outer tag, or for a structural component of that. In this case, the formal structure of the outer tag's content before unpacking usually no longer fulfills the validity conditions of the outer tag.

The underlying problem is not unique to Packed CBOR. For instance, [RFC8746] describes tags 64..87 that "stand in" for CBOR arrays (the native form of which has major type 4). For the other tags defined in this specification, which require some array structure of the tag content, a footnote was added:

> [...] The second element of the outer array in the data item is a native CBOR array (major type 4) or Typed Array (one of tag 64..87)

The top-down approach to handle the "rendezvous" between the outer and inner tags does not support extensibility: any further Typed Array tags being defined do not inherit the exception granted to tag number 64..87; they would need to formally update all existing tag definitions that can accept typed arrays or be of limited use with these existing tags.

Instead, the tag validity mechanism needs to be extended by a bottom-up component: A tag definition needs to be able to declare that the tag can "stand in" for, (is, in terms of tag validity, equivalent to) some structure.

E.g., tag 64..87 could have declared their equivalence to the CBOR major type 4 arrays they stand in for.

> Note that not all domain extensions to tags can be addressed using the equivalence principle: E.g., on a data model level, numbers with arbitrary exponents ([ARB-EXP], tags 264 and 265) are strictly a superset of CBOR's predefined fractional types, tags 4 and 5. They could not simply declare that they are equivalent to tags 4 and 5 as a tag requiring a fractional value may have no way to handle the extended range of tag 264 and 265.

5.1. Tag Equivalence

A tag definition **MAY** declare Tag Equivalence to some existing structure for the tag, under some conditions defined by the new tag definition. This, in effect, extends all existing tag definitions that accept the named structure to accept the newly defined tag under the conditions given for the Tag Equivalence.

A number of limitations apply to Tag Equivalence, which therefore should be applied deliberately and sparingly:

* Tag Equivalence is a new concept, which may not be implemented by an existing generic decoder. A generic decoder not implementing tag equivalence might raise tag validity errors where Tag Equivalence says there should be none.

* A CBOR protocol **MAY** specify the use of Tag Equivalence, effectively limiting its full use to those generic encoders that implement it. Existing CBOR protocols that do not address Tag Equivalence implicitly have a new variant that allows Tag Equivalence (e.g., to support Packed CBOR with an existing protocol). A CBOR protocol that does address Tag Equivalence **MAY** be explicit about what kinds of Tag Equivalence it supports (e.g., only the reference tags employed by Packed CBOR and certain table setup tags).

* There is currently no way to express Tag Equivalence in CDDL. For Packed CBOR, CDDL would typically be used to describe the unpacked CBOR represented by it; further restricting the Packed CBOR is likely to lead to interoperability problems. (Note that, by definition, there is no need to describe Tag Equivalence on the receptacle side; only for the tag that declares Tag Equivalence.)

* The registry "CBOR Tags" [IANA.cbor-tags] currently does not have a way to record any equivalence claimed for a tag. A convention would be to alert to Tag Equivalence in the "Semantics (short

form)" field of the registry.Needs to be done for the tag registrations here.

5.2. Tag Equivalence of Packed CBOR Tags

The reference tags in this specification declare their equivalence to the unpacked shared items or function results they represent.

The table setup tag 113 declares its equivalence to the unpacked CBOR data item represented by it.

6. IANA Considerations

6.1. CBOR Tags Registry

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, tag (for straight)	Packed CBOR: shared/ straight	draft-ietf-cbor-packed
105	text string, byte string, array, map, tag	Packed CBOR: ijoin function	draft-ietf-cbor-packed
106	text string, byte string, array, map, tag	Packed CBOR: join function	draft-ietf-cbor-packed
113	array (shared-items, argument-items, rump)	Packed CBOR: table setup	draft-ietf-cbor-packed
224-255	text string, byte string, array, map, tag	Packed CBOR: straight	draft-ietf-cbor-packed
28704-32767	text string, byte string, array, map, tag	Packed CBOR: straight	draft-ietf-cbor-packed
1879052288-2147483647	text string, byte string, array, map, tag	Packed CBOR: straight	draft-ietf-cbor-packed
216-223	text string, byte string, array, map, tag	Packed CBOR: inverted	draft-ietf-cbor-packed
27647-28671	text string, byte string, array, map, tag	Packed CBOR: inverted	draft-ietf-cbor-packed
1811940352-1879048191	text string, byte string, array, map, tag	Packed CBOR: inverted	draft-ietf-cbor-packed

Table 4: Values for Tag Numbers

6.2. CBOR Simple Values Registry

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

7. Security Considerations

The security considerations of [STD94] apply.

Loops in the Packed CBOR can be used as a denial of service attack, see Section 2.5.

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.)

8. References

8.1. Normative References

[IANA.cbor-simple-values] IANA, "Concise Binary Object Representation (CBOR) Simple Values", 19 September 2013, <https://www.iana.org/assignments/cbor-simple-values>.

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", 19 September 2013, <https://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>.

[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>.

8.2. Informative References

[ARB-EXP]

Occil, P., "Arbitrary-Exponent Numbers", Specification for Registration of CBOR Tags 264 and 265, <http://peteroupc.github.io/CBOR/bigfrac.html>.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <https://www.rfc-editor.org/info/rfc1951>.

[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>.

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

[RFC8746] Bormann, C., Ed., "Concise Binary Object Representation (CBOR) Tags for Typed Arrays", RFC 8746, DOI 10.17487/RFC8746, February 2020, <https://www.rfc-editor.org/info/rfc8746>.

[STD63] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

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

```
113([["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

```
113([/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 /
   /argument/ ["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 /
   /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 [STD94]. 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 what turned into 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