ï»¿

Concise Binary Object Representation (CBOR) Tags for Typed Arrays

Abstract

   The Concise Binary Object Representation (CBOR), as defined in RFC
   7049, 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.

   This document makes use of this extensibility to define a number of
   CBOR tags for typed arrays of numeric data, as well as additional
   tags for multi-dimensional and homogeneous arrays. It is intended as
   the reference document for the IANA registration of the CBOR tags
   defined.

Status of This Memo

   This is an Internet Standards Track document.

   This document is a product of the Internet Engineering Task Force
   (IETF). It represents the consensus of the IETF community. It has
   received public review and has been approved for publication by the
   Internet Engineering Steering Group (IESG). Further information on
   Internet Standards is available in Section 2 of RFC 7841.

   Information about the current status of this document, any errata,
   and how to provide feedback on it may be obtained at
   https://www.rfc-editor.org/info/rfc8746.

Copyright Notice

Table of Contents

1. Introduction

      The Concise Binary Object Representation (CBOR) [RFC7049] provides
      for the interchange of structured data without a requirement for a
      pre-agreed schema. [RFC7049] defines a basic set of data types as
      well as a tagging mechanism that enables extending the set of data
      types supported via an IANA registry.

      Recently, a simple form of typed arrays of numeric data has received
      interest both in the Web graphics community [TypedArray] and in the
      JavaScript specification (see Section 22.2 (https://www.ecma-
      international.org/ecma-262/10.0/index.html#sec-typedarray-objects) of
      [ECMA-ES10]) as well as in corresponding implementations
      [ArrayBuffer].

      Since these typed arrays may carry significant amounts of data, there
      is interest in interchanging them in CBOR without the need of lengthy
      conversion of each number in the array. This can also save space
      overhead with encoding a type for each element of an array.

      This document defines a number of interrelated CBOR tags that cover
      these typed arrays, as well as additional tags for multi-dimensional
      and homogeneous arrays. It is intended as the reference document for
      the IANA registration of the tags defined.

      Note that an application that generates CBOR with these tags has
      considerable freedom in choosing variants (e.g., with respect to
      endianness, embedded type (signed vs. unsigned), and number of bits
      per element) or whether a tag defined in this specification is used
      at all instead of more basic CBOR. In contrast to representation
      variants of single CBOR numbers, there is no representation that
      could be identified as "preferred". If deterministic encoding is
      desired in a CBOR-based protocol making use of these tags, the
      protocol has to define which of the encoding variants are used for
      each individual case.

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.

      The term "byte" is used in its now-customary sense as a synonym for
      "octet". Where bit arithmetic is explained, this document uses
      familiar notation from the programming language C [C] (including
      C++14's 0bnnn binary literals [CPlusPlus]) with the exception of the
      operator "**", which stands for exponentiation.

      The term "array" is used in a general sense in this document unless
      further specified. The term "classical CBOR array" describes an
      array represented with CBOR major type 4. A "homogeneous array" is
      an array of elements that are all the same type (the term is neutral
      as to whether that is a representation type or an application data
      model type).

      The terms "big endian" and "little endian" are used to indicate a
      most significant byte first (MSB first) representation of integers
      and a least significant byte first (LSB first) representation,
      respectively.

2. Typed Arrays

      Typed arrays are homogeneous arrays of numbers, all of which are
      encoded in a single form of binary representation. The concatenation

of these representations is encoded as a single CBOR byte string
(major type 2), enclosed by a single tag indicating the type and
encoding of all the numbers represented in the byte string.

2.1. Types of Numbers

Three classes of numbers are of interest: unsigned integers (uint),
signed integers (two's complement, sint), and IEEE 754 binary
floating point numbers (which are always signed). For each of these
classes, there are multiple representation lengths in active use:

Length ll	uint	sint	float
===========	========	========	===========
0	uint8	sint8	binary16
1	uint16	sint16	binary32
2	uint32	sint32	binary64
3	uint64	sint64	binary128

                       Table 1: Length Values

Here, sintN stands for a signed integer of exactly N bits (for
instance, sint16), and uintN stands for an unsigned integer of
exactly N bits (for instance, uint32). The name binaryN stands for
the number form of the same name defined in IEEE 754 [IEEE754].

Since one objective of these tags is to be able to directly ship the
ArrayBuffers underlying the Typed Arrays without re-encoding them,
and these may be either in big-endian (network byte order) or in
little-endian form, we need to define tags for both variants.

In total, this leads to 24 variants. In the tag, we need to express
the choice between integer and floating point, the signedness (for
integers), the endianness, and one of the four length values.

In order to simplify implementation, a range of tags is being
allocated that allows retrieving all this information from the bits
of the tag: tag values from 64 to 87.

The value is split up into 5 bit fields: 0b010, f, s, e, and ll as
detailed in Table 2.

Field	Use
=======	===
0b010	the constant bits 0, 1, 0
f	0 for integer, 1 for float
s	0 for float or unsigned integer, 1 for signed integer
e	0 for big endian, 1 for little endian
ll	A number for the length (Table 1).

             Table 2: Bit Fields in the Low 8 Bits of the Tag

The number of bytes in each array element can then be calculated by
"2**(f + ll)" (or "1 << (f + ll)" in a typical programming language).
(Notice that 0f and ll are the two least significant bits,
respectively, of each 4-bit nibble in the byte.)

In the CBOR representation, the total number of elements in the array
is not expressed explicitly but is implied from the length of the
byte string and the length of each representation. It can be
computed from the length, in bytes, of the byte string comprising the

representation of the array by inverting the previous formula:
"bytelength >> (f + ll)".

For the uint8/sint8 values, the endianness is redundant. Only the
tag for the big-endian variant is used and assigned as such. The tag
that would signify the little-endian variant of sint8 MUST NOT be
used; its tag number is marked as reserved. As a special case, the
tag that would signify the little-endian variant of uint8 is instead
assigned to signify that the numbers in the array are using clamped
conversion from integers, as described in more detail in
Section 7.1.11 (http://www.ecma-international.org/ecma-262/6.0/#sec-
touint8clamp) of the ES10 JavaScript specification ("ToUint8Clamp")
[ECMA-ES10]; the assumption here is that a program-internal
representation of this array after decoding would be marked this way
for further processing providing "roundtripping" of JavaScript-typed
arrays through CBOR.

IEEE 754 binary floating numbers are always signed. Therefore, for
the float variants ("f" == 1), there is no need to distinguish
between signed and unsigned variants; the "s" bit is always zero.
The tag numbers where "s" would be one (which would have tag values
88 to 95) remain free to use by other specifications.

3. Additional Array Tags

   This specification defines three additional array tags. The Multi-
   dimensional Array tags can be combined with classical CBOR arrays as
   well as with Typed Arrays in order to build multi-dimensional arrays
   with constant numbers of elements in the sub-arrays. The Homogeneous
   Array tag can be used as a signal by an application to identify a
   classical CBOR array as a homogeneous array, even when a Typed Array
   does not apply.

3.1. Multi-dimensional Array

   A multi-dimensional array is represented as a tagged array that
   contains two (one-dimensional) arrays. The first array defines the
   dimensions of the multi-dimensional array (in the sequence of outer
   dimensions towards inner dimensions) while the second array
   represents the contents of the multi-dimensional array. If the
   second array is itself tagged as a Typed Array, then the element type
   of the multi-dimensional array is known to be the same type as that
   of the Typed Array.

   Two tags are defined by this document: one for elements arranged in
   row-major order and another for column-major order [RowColMajor].

3.1.1. Row-Major Order

   Tag: 40

   Data Item: Array (major type 4) of two arrays: one array (major type
      4) of dimensions, which are unsigned integers distinct from zero;
      and one array (any one of a CBOR array of major type 4, a Typed
      Array, or a Homogeneous Array) of elements.

   Data in the second array consists of consecutive values where the
   last dimension is considered contiguous (row-major order).

   Figure 1 shows a declaration of a two-dimensional array in the C
   language, a representation of that in CBOR using both a multi-
   dimensional array tag and a typed array tag.

   uint16_t a[2][3] = {
     {2, 4, 8},   /* row 0 */
     {4, 16, 256},
   };

   <Tag 40> # multi-dimensional array tag
      82   # array(2)
        82   # array(2)

```
          02     # unsigned(2) 1st Dimension
          03     # unsigned(3) 2nd Dimension
       <Tag 65> # uint16 array
          4c     # byte string(12)
            0002 # unsigned(2)
            0004 # unsigned(4)
            0008 # unsigned(8)
            0004 # unsigned(4)
            0010 # unsigned(16)
            0100 # unsigned(256)
```

            Figure 1: Multi-dimensional Array in C and CBOR

   Figure 2 shows the same two-dimensional array using the multi-
   dimensional array tag in conjunction with a basic CBOR array (which,
   with the small numbers chosen for the example, happens to be
   shorter).

```
   <Tag 40> # multi-dimensional array tag
      82        # array(2)
         82        # array(2)
            02     # unsigned(2) 1st Dimension
            03     # unsigned(3) 2nd Dimension
         86     # array(6)
            02       # unsigned(2)
            04       # unsigned(4)
            08       # unsigned(8)
            04       # unsigned(4)
            10       # unsigned(16)
            19 0100 # unsigned(256)
```

           Figure 2: Multi-dimensional Array Using Basic CBOR Array

3.1.2. Column-Major Order

   The multi-dimensional arrays specified in the previous sub-subsection
   are in "row major" order, which is the preferred order for the
   purposes of this specification. An analogous representation that
   uses "column major" order arrays is provided in this subsection under
   the tag 1040, as illustrated in Figure 3.

   Tag: 1040

   Data Item: The same as tag 40, except the data in the second array
      consists of consecutive values where the first dimension is
      considered contiguous (column-major order).

```
   <Tag 1040> # multi-dimensional array tag, column-major order
      82        # array(2)
         82        # array(2)
            02     # unsigned(2) 1st Dimension
            03     # unsigned(3) 2nd Dimension
         86     # array(6)
            02       # unsigned(2)
            04       # unsigned(4)
            04       # unsigned(4)
            10       # unsigned(16)
            08       # unsigned(8)
            19 0100 # unsigned(256)
```

      Figure 3: Multi-dimensional Array Using Basic CBOR Array, Column-
                              Major Order

3.2. Homogeneous Array

   Tag: 41

   Data Item: Array (major type 4)

   This tag identifies the classical CBOR array (a one-dimensional
   array) tagged by it as a homogeneous array, that is, it has elements

that are all of the same application model data type. The element
type of the array is therefore determined by the application model
data type of the first array element.

This can be used in application data models that apply specific
semantics to homogeneous arrays. Also, in certain cases,
implementations in strongly typed languages may be able to create
native homogeneous arrays of specific types instead of ordered lists
while decoding. Which CBOR data items constitute elements of the
same application type is specific to the application.

Figure 4 shows an example for a homogeneous array of booleans in C++
[CPlusPlus] and CBOR.

```
bool boolArray[2] = { true, false };

<Tag 41>  # Homogeneous Array Tag
   82           #array(2)
      F5        # true
      F4        # false
```

              Figure 4: Homogeneous Array in C++ and CBOR

Figure 5 extends the example with a more complex structure.

```
typedef struct {
  bool active;
  int value;
} foo;
foo myArray[2] = { {true, 3}, {true, -4} };

<Tag 41>
    82  # array(2)
       82  #  array(2)
             F5  # true
             03  # 3
       82 # array(2)
             F5  # true
             23  # -4
```

              Figure 5: Homogeneous Array in C++ and CBOR

4. Discussion

   Support for both little- and big-endian representation may seem out
   of character with CBOR, which is otherwise fully big endian. This
   support is in line with the intended use of the typed arrays and the
   objective not to require conversion of each array element.

   This specification allocates a sizable chunk out of the single-byte
   tag space. This use of code point space is justified by the wide use
   of typed arrays in data interchange.

   Providing a column-major order variant of the multi-dimensional array
   may seem superfluous to some and useful to others. It is cheap to
   define the additional tag so that it is available when actually
   needed. Allocating it out of a different number space makes the
   preference for row-major evident.

   Applying a Homogeneous Array tag to a Typed Array would usually be
   redundant and is therefore not provided by the present specification.

5. CDDL Typenames

   For use with CDDL [RFC8610], the typenames defined in Figure 6 are
   recommended:

```
ta-uint8 = #6.64(bstr)
ta-uint16be = #6.65(bstr)
ta-uint32be = #6.66(bstr)
```

```
ta-uint64be = #6.67(bstr)
ta-uint8-clamped = #6.68(bstr)
ta-uint16le = #6.69(bstr)
ta-uint32le = #6.70(bstr)
ta-uint64le = #6.71(bstr)
ta-sint8 = #6.72(bstr)
ta-sint16be = #6.73(bstr)
ta-sint32be = #6.74(bstr)
ta-sint64be = #6.75(bstr)
; reserved: #6.76(bstr)
ta-sint16le = #6.77(bstr)
ta-sint32le = #6.78(bstr)
ta-sint64le = #6.79(bstr)
ta-float16be = #6.80(bstr)
ta-float32be = #6.81(bstr)
ta-float64be = #6.82(bstr)
ta-float128be = #6.83(bstr)
ta-float16le = #6.84(bstr)
ta-float32le = #6.85(bstr)
ta-float64le = #6.86(bstr)
ta-float128le = #6.87(bstr)
homogeneous<array> = #6.41(array)
multi-dim<dim, array> = #6.40([dim, array])
multi-dim-column-major<dim, array> = #6.1040([dim, array])
```

                    Figure 6: Recommended Typenames for CDDL

6. IANA Considerations

    IANA has allocated the tags in Table 3 using this document as the
    specification reference. (The reserved value is for a future
    revision of typed array tags.)

    The allocations were assigned from the "specification required" space
    (24..255) with the exception of 1040, which was assigned from the
    "first come first served" space (256..).

Tag	Data Item	Semantics
40	array of two arrays*	Multi-dimensional Array, row-major order
41	array	Homogeneous Array
64	byte string	uint8 Typed Array
65	byte string	uint16, big endian, Typed Array
66	byte string	uint32, big endian, Typed Array
67	byte string	uint64, big endian, Typed Array
68	byte string	uint8 Typed Array, clamped arithmetic
69	byte string	uint16, little endian, Typed Array
70	byte string	uint32, little endian, Typed Array
71	byte string	uint64, little endian, Typed Array
72	byte string	sint8 Typed Array

73	byte string	sint16, big endian, Typed Array
74	byte string	sint32, big endian, Typed Array
75	byte string	sint64, big endian, Typed Array
76	byte string	(reserved)
77	byte string	sint16, little endian, Typed Array
78	byte string	sint32, little endian, Typed Array
79	byte string	sint64, little endian, Typed Array
80	byte string	IEEE 754 binary16, big endian, Typed Array
81	byte string	IEEE 754 binary32, big endian, Typed Array
82	byte string	IEEE 754 binary64, big endian, Typed Array
83	byte string	IEEE 754 binary128, big endian, Typed Array
84	byte string	IEEE 754 binary16, little endian, Typed Array
85	byte string	IEEE 754 binary32, little endian, Typed Array
86	byte string	IEEE 754 binary64, little endian, Typed Array
87	byte string	IEEE 754 binary128, little endian, Typed Array
1040	array of two arrays*	Multi-dimensional Array, column-major order

Table 3: Values for Tags

*40 or 1040 data item: 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)

7. Security Considerations

   The security considerations of [RFC7049] apply; special attention is
   drawn to the second paragraph of Section 8 of [RFC7049].

   The tag for homogeneous arrays makes a promise about its tagged data
   item, which a maliciously constructed CBOR input can then choose to
   ignore. As always, the decoder therefore has to ensure that it is
   not driven into an undefined state by array elements that do not
   fulfill the promise, and that it does continue to fulfill its API
   contract in this case as well.

   As with all formats that are used for data interchange, an attacker
   may have control over the shape of the data delivered as input to the
   application, which therefore needs to validate that shape before it
   makes it the basis of its further processing. One unique aspect that
   typed arrays add to this is that an attacker might substitute a

Uint8ClampedArray for where the application expects a Uint8Array, or vice versa, potentially leading to very different (and unexpected) processing semantics of the in-memory data structures constructed. Applications that could be affected by this will therefore need to be careful about making this distinction in their input validation.

8. References

8.1. Normative References

[C]   International Organization for Standardization, "Information technology – Programming languages – C", ISO/IEC 9899:2018, Fourth Edition, June 2018.

[CPlusPlus]
             International Organization for Standardization, "Programming languages – C++", ISO/IEC 14882:2017, Fifth Edition, December 2017.

[ECMA-ES10]
             ECMA International, "ECMAScript 2019 Language Specification", Standard ECMA-262 10th Edition, June 2019, <https://www.ecma-international.org/ecma-262/10.0/index.html>.

[IEEE754]   IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2019, DOI 10.1109/IEEESTD.2019.8766229, <https://ieeexplore.ieee.org/document/8766229>.

[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>.

[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>.

[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>.

8.2. Informative References

[ArrayBuffer]
             Mozilla Developer Network, "JavaScript typed arrays", June 2010, <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays>.

[RowColMajor]
             Wikipedia, "Row- and column-major order", September 2019, <https://en.wikipedia.org/w/index.php?title=Row-_and_column-major_order&oldid=917905325>.

[TypedArray]
             Vukicevic, V. and K. Russell, "Typed Array Specification", February 2011, <https://web.archive.org/web/20110207024413/http://www.khronos.org/registry/typedarray/specs/latest/>.

Acknowledgements

Contributors

Author's Address

Carsten Bormann (editor)
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany

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