Concise Binary Object Representation (CBOR) Tags for Typed Arrays
draft-ietf-cbor-array-tags-03

Abstract

The Concise Binary Object Representation (CBOR, 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.

The present document makes use of this extensibility to define a number of CBOR tags for typed arrays of numeric data, as well as two 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 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 September 6, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of
1. Introduction

The Concise Binary Object Representation (CBOR, [RFC7049]) provides the interchange of structured data without a requirement for a pre-agreed schema. RFC 7049 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 have received interest both in the Web graphics community [TypedArray] and in the JavaScript specification [TypedArrayES6], 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 document defines a number of interrelated CBOR tags that cover these typed arrays, as well as two additional tags for multi-dimensional and homogeneous arrays. It is intended as the reference document for the IANA registration of the tags defined.
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 the notation familiar from the programming language C (including C++14’s 0bnnn binary literals), except that the operator "***" stands for exponentiation.

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:

<table>
<thead>
<tr>
<th>Length ll</th>
<th>uint</th>
<th>sint</th>
<th>float</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>uint8</td>
<td>sint8</td>
<td>binary16</td>
</tr>
<tr>
<td>1</td>
<td>uint16</td>
<td>sint16</td>
<td>binary32</td>
</tr>
<tr>
<td>2</td>
<td>uint32</td>
<td>sint32</td>
<td>binary64</td>
</tr>
<tr>
<td>3</td>
<td>uint64</td>
<td>sint64</td>
<td>binary128</td>
</tr>
</tbody>
</table>

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.

Since one objective of these tags is to be able to directly ship the ArrayBuffer 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_ll, as detailed in Table 2.

<table>
<thead>
<tr>
<th>Field</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b010</td>
<td>the constant bits 0, 1, 0</td>
</tr>
<tr>
<td>f</td>
<td>0 for integer, 1 for float</td>
</tr>
<tr>
<td>s</td>
<td>0 for unsigned integer or float, 1 for signed integer</td>
</tr>
<tr>
<td>e</td>
<td>0 for big endian, 1 for little endian</td>
</tr>
<tr>
<td>ll</td>
<td>A number for the length (Table 1).</td>
</tr>
</tbody>
</table>

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).

In the CBOR representation, the total number of elements in the array is not expressed explicitly, but implied from the length of the byte string and the length of each representation. It can be computed inversely to the previous formula from the length of the byte string in bytes: "bytelength >> (f + ll)".

For the uint8/sint8 values, the endianness is redundant. Only the big endian variant is used. The little endian variant of sint8 MUST NOT be used, its tag is marked as reserved. As a special case, the tag number that would have been the little endian variant of uint8 is used to signify that the numbers in the array are using clamped conversion from integers, as described in more detail in Section 7.1 of [TypedArrayUpdate].
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 to facilitate the ingestion of homogeneous classical CBOR arrays, providing performance advantages even when a Typed Array does not apply.

3.1. Multi-dimensional Array

Tag: 40

Data Item: array (major type 4) of two arrays, one array (major type 4) of dimensions, and one array (major type 4, a Typed Array, or a Homogeneous Array) of elements

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. Data in the Typed Array byte string 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 multidimensional 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 multidimensional 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

Tag: 1040

Data Item: as with tag 40

Note that above arrays 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
under the tag 1040, as illustrated in Figure 3.

<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 provides a hint to decoders that the array tagged by it has
elements that are all of the same application type. The element type
of the array is thus determined by the application type of the first
array element. This can be used by implementations in strongly typed
languages while decoding to create native homogeneous arrays of
specific types instead of ordered lists.

Which CBOR data items constitute elements of the same application
type is specific to the application. However, type systems of
programming languages have enough commonality that an application
should be able to create portable homogeneous arrays.

Figure 4 shows an example for a homogeneous array of booleans in C++
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 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 be redundant and is therefore not provided by the present specification.
5. CDDL typenames

For the use with CDDL [I-D.ietf-cbor-cddl], the typenames defined in Figure 6 are recommended:

\begin{verbatim}
  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])
\end{verbatim}

Figure 6: Recommended typenames for CDDL
6. IANA Considerations

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

The allocations came out of the "specification required" space (24..255), with the exception of 1040, which came out of the "first come first served" space (256..).
<table>
<thead>
<tr>
<th>Tag</th>
<th>Data Item</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>byte string</td>
<td>uint8 Typed Array</td>
</tr>
<tr>
<td>65</td>
<td>byte string</td>
<td>uint16, big endian, Typed Array</td>
</tr>
<tr>
<td>66</td>
<td>byte string</td>
<td>uint32, big endian, Typed Array</td>
</tr>
<tr>
<td>67</td>
<td>byte string</td>
<td>uint64, big endian, Typed Array</td>
</tr>
<tr>
<td>68</td>
<td>byte string</td>
<td>uint8 Typed Array, clamped arithmetic</td>
</tr>
<tr>
<td>69</td>
<td>byte string</td>
<td>uint16, little endian, Typed Array</td>
</tr>
<tr>
<td>70</td>
<td>byte string</td>
<td>uint32, little endian, Typed Array</td>
</tr>
<tr>
<td>71</td>
<td>byte string</td>
<td>uint64, little endian, Typed Array</td>
</tr>
<tr>
<td>72</td>
<td>byte string</td>
<td>sint8 Typed Array</td>
</tr>
<tr>
<td>73</td>
<td>byte string</td>
<td>sint16, big endian, Typed Array</td>
</tr>
<tr>
<td>74</td>
<td>byte string</td>
<td>sint32, big endian, Typed Array</td>
</tr>
<tr>
<td>75</td>
<td>byte string</td>
<td>sint64, big endian, Typed Array</td>
</tr>
<tr>
<td>76</td>
<td>byte string</td>
<td>(reserved)</td>
</tr>
<tr>
<td>77</td>
<td>byte string</td>
<td>sint16, little endian, Typed Array</td>
</tr>
<tr>
<td>78</td>
<td>byte string</td>
<td>sint32, little endian, Typed Array</td>
</tr>
<tr>
<td>79</td>
<td>byte string</td>
<td>sint64, little endian, Typed Array</td>
</tr>
<tr>
<td>80</td>
<td>byte string</td>
<td>IEEE 754 binary16, big endian, Typed Array</td>
</tr>
<tr>
<td>81</td>
<td>byte string</td>
<td>IEEE 754 binary32, big endian, Typed Array</td>
</tr>
<tr>
<td>82</td>
<td>byte string</td>
<td>IEEE 754 binary64, big endian, Typed Array</td>
</tr>
<tr>
<td>83</td>
<td>byte string</td>
<td>IEEE 754 binary128, big endian, Typed Array</td>
</tr>
<tr>
<td>84</td>
<td>byte string</td>
<td>IEEE 754 binary16, little endian, Typed Array</td>
</tr>
<tr>
<td>85</td>
<td>byte string</td>
<td>IEEE 754 binary32, little endian, Typed Array</td>
</tr>
<tr>
<td>86</td>
<td>byte string</td>
<td>IEEE 754 binary64, little endian, Typed Array</td>
</tr>
<tr>
<td>87</td>
<td>byte string</td>
<td>IEEE 754 binary128, little endian, Typed Array</td>
</tr>
<tr>
<td>40</td>
<td>array of two</td>
<td>Multi-dimensional Array, row-major order</td>
</tr>
<tr>
<td>1040</td>
<td>array of two</td>
<td>Multi-dimensional Array, column-major order</td>
</tr>
<tr>
<td>41</td>
<td>array</td>
<td>Homogeneous Array</td>
</tr>
</tbody>
</table>

Table 3: Values for Tags

*) 40 or 1040 data item: second element of outer array in data item is native CBOR array (major type 4) or Typed Array (one of Tag 64..87)
7. Security Considerations

The security considerations of RFC 7049 apply; special attention is drawn to the second paragraph of Section 8 of RFC 7049. The tags introduced here are not expected to raise security considerations beyond those.
8. References

8.1. Normative References

[I-D.ietf-cbor-cddl]


8.2. Informative References

[ArrayBuffer]

[TypedArray]

[TypedArrayES6]

[TypedArrayUpdate]
Contributors

Glenn Engel suggested the tags for multi-dimensional arrays and homogeneous arrays.

Acknowledgements

Jim Schaad reminded us that column-major order still is in use. IANA helped correct an error in a previous version.

Authors’ Addresses

Johnathan Roatch
Email: jroatch@gmail.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

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