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. 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 http://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 10, 2015.

Copyright Notice

Copyright (c) 2015 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 (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents.
1.  Introduction

   The Concise Binary Object Representation (CBOR, [RFC7049]) provides for
   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] and 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.  It is intended as the reference document for the
   IANA registration of the tags defined.

1.1.  Terminology

   The term "byte" is used in its now customary sense as a synonym for
   "octet".
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 (twos’ 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.

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 (0x40 to 0x57).

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>a constant '010'</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).
(Notice that f and ll are the lsb of each nibble (4bit) in the byte.)

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: "bytelength >> (f + ll)".

For the uint8/sint8 values, the endianness is redundant. Only the
big endian variant is used. As a special case, what would be the
little endian variant of uint8 is used to signify that the numbers in
the array are using clamped conversion from integers, as defined in
Section 7.1 of [TypedArrayUpdate].

3. 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.
4. CDDL typenames

For the use with CDDL [I-D.greevenbosch-appsawg-cbor-cddl], the typenames defined in Figure 1 are recommended:

```plaintext
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)
```

Figure 1: Recommended typenames for CDDL
5. IANA Considerations

IANA is requested to reserve the tags in Table 3, with the present document as the specification reference.

+-----+-------------+-----------------------------------------------+
<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>
</tbody>
</table>
+-----+-------------+-----------------------------------------------+

Table 3: Values for Tags

6. Security Considerations

The security considerations of RFC 7049 apply; the tags introduced here are not expected to raise security considerations beyond those.
7. References

7.1. Normative References

[I-D.greevenbosch-appsawg-cbor-cddl]


7.2. Informative References


Authors’ Addresses

Johnathan Roatch

Email: jroatch@gmail.com