Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 18, 2014

B. Greevenbosch
Huawei Technologies

June 27, 2014

CBOR data definition language: a notational convention to express CBOR data structures. draft-greevenbosch-appsawg-cbor-cddl-02

Abstract

This document proposes a notational convention to express CBOR data structures. Its main goal is to make it easy to express message structures for protocols that use 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 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 December 27, 2014.

Copyright Notice

Table of Contents

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

In this document, a notational convention to express CBOR [RFC7049] data structures is defined.

The main goal for the convention is to provide a unified notation that can be used when defining protocols that use CBOR.

The CBOR notational convention has the following goals:

(G1) Able to provide an unambiguous description of a CBOR data structures.

(G2) Easy for humans to read and write.

(G3) Flexibility to express the freedoms of choice in the CBOR data format.

(G4) Possibility to restrict format choices where appropriate.

(G5) Able to express common CBOR datatypes and structures.

(G6) Human and machine readable and processable.

(G7) Usable for automatic verification of whether CBOR data is compliant to a predefined format.

3. Definitions

The following contains a list of used words in this document:

"datatype" defines the format of a variable.

"variable" a data component encoded in CBOR.

4. Notational conventions

4.1. General conventions

The basic syntax is as follows:

- Each field has a name and a datatype.
- The name is written first, followed by a colon and then the datatype. The declarations is finished with a semicolon. Whitespace may appear around the colon and semicolon, as well as in front of the name.
- The name does not appear in the actual CBOR encoding.
- Comments are preceded by a '#' character.
- Variable names and datatypes are case sensitive.

4.2. Keywords for primitive datatypes

The following keywords for primitive datatypes are defined:

"bool" Boolean value (major type 7, additional information 20 or 21).

"bstr" A byte string (major type 2).

"float(16)" IEEE 754 half-precision float (major type 7, additional information 25).

"float(32)" IEEE 754 single-precision float (major type 7, additional information 26).

"float(64)" IEEE 754 double-precision float (major type 7, additional information 27).

"int" An unsigned integer (major type 0) or a negative integer (major type 1).

"nint" A negative integer (major type 1).

"simple" Simple value (major type 7, additional information 24).

"tstr" Text string (major type 3)

"uint" An unsigned integer (major type 0).

In addition, Section 4.6 defines datatypes associated with CBOR tags.

4.3. Arrays

Arrays can be of fixed length or of variable length. Both fixed length and variable length arrays can be implemented as definite and indefinite length arrays.

A fixed length array is is indicated by '[' and ']' characters behind its type, where number in between specifies the number of elements.

A variable length array can be indicated with a "*" behind its type.

The following is an example of an array of 4 integers:

```
fourNumbers: int[4];
```

The following is an example of a variable length array:

```
fibonacci : uint*;
```

4.4. Structures

Structures are a logical grouping of CBOR fields.

A structure has a name, which can be used as a value type for other fields. The name is followed by a '{' character and the definitions of the variables inside of the structure. The structure is closed by a '}' character.

A structure MAY be encoded as an array, in which case its name is preceded by a '*' character. Otherwise there is no CBOR encoding for the grouping.

The following is an example of a structure:

```
*Geography {
  city           : tstr;
  gpsCoordinates : GpsCoordinates;
}

GpsCoordinates {
  longitude      : uint;            # multiplied by 10^7
  lattitude      : uint;            # multiplied by 10^7
}
```

When encoding, the Geography structure is encoded using a CBOR array, whereas the GpsCoordinates do not have their own encompassing array.

4.5. Maps

If an entity is a map (major type 5), its datatype has the form

```
map( x, y )
```

where the keys have datatype x, and the values a datatype y.

If either x or y is unspecified (i.e. free to choose per entry), it is replaced by a '.'.

It is also possible to define a map with predefined keys as a type. In this case, type declaration is as follows:

```
x: map( y ) {
  key1: type1;
  key2: type2;
  ...
}
```

y is the datatype of the keys, and type1, type2, etc the datatype of the value associated with keys key1, key2 etc.

The name of an optional map element is preceded by a '?' character.

The example below the defines a map with display name (as a string), optionally the name components first name and family name (see Section 4.7 for more on optional variables), and age information (as an unsigned int).

```
PersonalData: map( tstr ) {
  "displayName": tstr;
  ?"nameComponents": NameComponents;
  "age": uint;
}

NameComponents: map( tstr, tstr ) {
  "firstName": tstr;
  "familyName" : tstr;
}
```

It is up to the application how to handle unknown tags, however, it is RECOMMENDED to ignore them.

4.6. Tags

A variable can have an associated CBOR tag (major type 6). This is indicated by the tag encapsulated between the square brackets '[' and ']', just before the variable's datatype definition.

For example, the following defines a positive bignum N:

```
N: [2]bstr;
```

[RFC7049] defines several tags. These tags can be also written using the datatypes from Table 1. For table rows with an empty "possible tag notation" entry, we refer to Table 3 in [RFC7049] and associated references for the possible encodings.

For example, the following is another way to define the bignum:

```
N: bignum;
```

datatype	possible tag notation	description
b64	[34]tstr	Base 64 (tag 34)
b64url	[33]tstr	Base 64 URL (tag 33)
bigfloat		bigfloat (tag 5)
bignum	[2]bstr or [3]bstr	positive (tag 2) or negative (tag 3) bignum
cbor	[24]bstr	Encoded CBOR data item (tag 24)
decfrac		decimal fraction (tag 4)
eb16		Expected conversion to base16 encoding (tag 23)
eb64		Expected conversion to base64 encoding (tag 22)
eb64url		Expected conversion to base64 url encoding (tag 21)
epochdt		epoch date/time (tag 1)
mime	[36]tstr	Mime message (tag 36)
nbignum	[3]bstr	negative bignum (tag 3)
regex	[35]tstr	regular expression (tag 35)
standarddt	[0]tstr	standard date/time string (tag 0)
ubignum	[2]bstr	positive bignum (tag 2)
uri	[32]tstr	URI (tag 32)

Table 1

4.7. Optional variables

There may be variables or structures whose inclusion is optional. In this case, the name of the variable is preceded by a '?' character

For example, the following defines a CBOR structure that is dependent on a boolean value.

```
*MainStruct {
  whichForm     : bool;
  ?data1        : Form1;   # when whichForm == true
  ?data2        : Form2;   # when whichForm == false
}

Form1 {
  anInteger     : int;
  aTextString   : tstr;
}

Form2 {
  aFloat        : float(16);
  aBinaryString : bstr;
}
```

Notice that it is not possible to define the relationship between "whichForm" and inclusion of either "data1" or "data2" with CBOR content rules. Such relationship should be otherwise communicated to the implementer, for example in the text of the specification that uses the CBOR structure, or with comments as was done in this example.

Protocol designers should exhibit utmost care when defining CBOR structures with optional variables, especially when some of these variables have the same datatype.

For example, the following CBOR data structure is ambiguous:

```
*DataStruct {
  ?OptionalVariable        : uint;
  MandatoryVariable        : uint;
  ?AnotherOptionalVariable : uint;
}
```

Since optional variables are often detected from their datatype, it is RECOMMENDED to not have a following of multiple variables of the same datatype, when some of these variables are optional.

5. Examples

This section contains various examples of structures defined using the CBOR notational convention.

5.1. Moves in a computer game

A multiplayer computer game uses CBOR to exchange moves between the players. To ensure a good gaming experience, the move information needs to be exchanged quickly and frequently. Therefore, the game uses CBOR to send its information in a compact format. Figure 1 shows definition of the CBOR information exchange format.

```
*UpdateMsg {
  move_no        : uint;                 # increases for each move
  player_info    : PlayerInfo;           # general information
  moves          : Moves*;               # moves in this message
}

PlayerInfo {
  alias          : tstr;
  player_id      : uint;
  experience     : uint;                 # beginner: 0; expert: 3
  gold           : uint;
  supplies       : Supplies;
  avg_strength   : float(16);
}

Supplies : map( uint ) {
  0              : uint;                 # wood
  1              : uint;                 # iron
  2              : uint;                 # grain
}

*Moves {
  unit_id        : uint;
  unit_strength  : uint;                 # between 0 and 100
  source_pos     : uint[2];              # (x,y)
  target_pos     : uint[2];              # (x,y)
}
```

Figure 1: CBOR definition of an information exchange format for a computer game

Notice that the supplies have been encoded as a map with integer keys. In this example, using string keys would also have been suitable. However, the example illustrates the possibility to use other datatypes for keys, leading to more efficient encoding.

Player "Johnny" does two moves. The game server has assigned Johnny the ID 0x7a3b871f. Johnny is an amateur player, so has experience 1. He currently has 1200 gold, 13 units of wood, 70 units of iron and 29

units of grain. He has several units, with a total average strength of 30.25.

The units Johnny plays in move 250 are the unit with ID 19, strength 20 from (5,7) to (6,9), and the unit with ID 87, strength 40 from (7,10) to (6,10).

This information is coded in CBOR as depicted in Figure 2.

```
9F
   18 FA                # move 250
   66 4A 6F 68 6E 6E 79 # "Johnny"
   1A 7A 3B 87 1F       # player_id
   01                   # experience
   19 04 B0             # 1200 gold as uint
   A3                   # begin map "supplies" with 3 elements
      00                # wood:
         0C             # 13 as uint
      01                # iron:
         18 86          # 70 as uint
      02                # grain:
         18 1D          # 29 as uint
   F9 4F 90             # average strength 30.25 half-precision float
   9F                   # indefinite length "moves" array
      84                # 4-element array Moves
         13             # unit id 19 as uint
         14             # strength 20 as uint
         82             # 2-element array source_pos
            05          # source_pos.x=5
            07          # source_pos.y=7
         82             # 2-element array target_pos
            06          # target_pos.x=6
            09          # target_pos.y=9
      84                # 4-element array Moves
         18 57          # unit id 87
         18 28          # strength 40
         82             # 2-element array source_pos
            07          # source_pos.x=7
            0a          # source_pos.y=10
         82             # 2-element array target_pos
            06          # target_pos.x=6
            0a          # target_pos.y=10
      FF                # end of "moves" array
   FF
```

Figure 2: CBOR instance for game example

5.2. Fruit

Figure 3 contains an example for a CBOR structure that contains information about fruit.

```
fruitlist              : Fruit*;

*Fruit {
  name                 : tstr;
  colour               : uint*;
  avg_weight           : float( 16 );
  price                : uint;
  international_names  : International;
  rfu                  : bstr;              # reserved for future use
}

International : map( tstr ) {
  "CN"                 : tstr;              # Chinese
  "NL"                 : tstr;              # Dutch
  "EN"                 : tstr;              # English
  "FR"                 : tstr;              # French
  "DE"                 : tstr;              # German
}
```

Figure 3: Example CBOR structure

The colour integer can have the values from Table 2.

Colour	Value
black	0
red	1
green	2
yellow	3
blue	4
magenta	5
cyan	6
white	7
orange	8
pink	9
purple	10
brown	11
grey	12

Table 2: Possible values for the colour field

For example, apples can be red, yellow or green. They have an average weight of 0.195kg and a price of 30 cents. Chinese for "apple" in UTF-8 is [E8 8B B9 E6 9E 9C], the Dutch word is "appel" and the French word "pomme".

For simplicity, let's assume that the colour of oranges can only be orange. They have an average weight of 0.230kg and a price of 50 cents. Chinese for "orange" in UTF-8 is [E6 A9 99 E5 AD 90], the Dutch word is "sinaasappel" and the German word "Orange".

This information would be encoded as depicted in Figure 4.

```
9F                              # indefinite length "fruitlist" array
   86                           # First "Fruit" instance, 6 elements
      65                        # text string "name" length 5
         61 70 70 6C 65         # "apple"
      83                        # array for "Colour", 3 elements
         01                     # "red" as uint
         02                     # "green" as uint
         03                     # "yellow" as uint
      F9                        # Floating point half precision
         32 3D                  # "avg_weight" 0.195
      18 1E                     # "price" 30 as uint
      A3                        # map "international_names", 3 pairs
         62 43 4E               # text string length 2, "CN"
         66 E8 8B B9 E6 9E 9C   # Chinese word for apple
         62 4E 4C               # "NL"
         65 61 70 70 65 6C      # "appel"
         62 46 52               # "FR"
         65 70 6F 6D 6D 65      # "pomme"
      40                        # byte string "rfu", 0 bytes length
   86                           # Second "Fruit" instance
      66                        # text string "name" length 6
         6F 72 61 6E 67 65      # "orange"
      81                        # array for "Colour", 3 elements
         08                     # "orange" as uint
      F9                        # Floating point half precision
         33 5C                  # "avg_weight" 0.230
      18 32                     # "price" 50 as uint
      A3                        # map "international_names", 3 pairs
         62 43 4E               # text string length 2, "CN"
         66 E6 A9 99 E5 AD 90   # Chinese word for orange
         62 4E 4C               # "NL"
         6B 73 69 6E 61 61 73 61 70 70 65 6C # "sinaasappel"
         62 44 45               # "DE"
         66 4F 72 61 6E 67 65   # "Orange"
      40                        # byte string "rfu", 0 bytes length
   FF                           # end of "fruitlist" array
```

Figure 4: Example CBOR instance

Notice that if the "Fruit" structure did not have the preceding "*", the two "Fruit" instance arrays would have been omitted. In addition, the "fruitlist" array would have had 12 elements instead of 2. (Although for "fruitlist" the indefinite length approach was chosen, such that the number of elements is not explicitely signalled.)

6. Philosophy

The CBOR notational convention can be used to efficiently define the layout of CBOR data.

In addition, it has been specified such that a machine can verify whether or not CBOR data is compliant to its definition. The thoroughness of this compliance verification depends on the application.

For example, an application may decide not to verify the data structure at all, and use the CBOR content rules solely as a means to indicate the structure of the data to the programmer.

On the other end, the application may also implement a verification method that goes as far as verifying that all mandatory map keys are available.

The matter in how far the data description must be enforced by an application is left to the implementers and specifiers of that application.

7. Open Issues

At least the following issues need further consideration:

- Whether to remove optional variables (other than in maps).
- More extensive security considerations.
- Consider whether structures without encapsulating CBOR array encoding should be omitted.

8. Change Log

Changes from version 00 to version 01

- Removed constants
- Updated the tag mechanism
- Extended the map structure
- Added examples

9. Security considerations

This document presents a content rules language for expressing CBOR data structures. As such, it does not bring any security issues on itself, although specification of protocols that use CBOR naturally need security analysis when defined.

10. IANA considerations

This document does not require any IANA registrations.

11. Acknowledgements

For this draft, there has been inspiration from the C and Pascal languages, MPEG's conventions for describing structures in the ISO base media file format, and Andrew Lee Newton's "JSON Content Rules" draft.

Useful feedback came from Carsten Bormann and Joe Hildebrand.

12. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

Author's Address

Bert Greevenbosch
Huawei Technologies Co., Ltd.
Huawei Industrial Base
Bantian, Longgang District
Shenzhen 518129
P.R. China

Email: bert.greevenbosch@huawei.com