Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 19, 2015

B. Greevenbosch
R. Sun
Huawei Technologies
C. Vigano
University of Bremen

December 16, 2014

CBOR data definition language: a notational convention to express CBOR data structures.
draft-greevenbosch-appsawg-cbor-cddl-04

Abstract

This document proposes a notational convention to express CBOR data structures. Its main goal is to provide an easy and unambiguous way to express structures for protocol messages and data formats that use CBOR.

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) 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) Automatic data format compliancy verification.
(G8) Extraction of specific elements from CBOR data for further processing.

This document has the following structure:

The syntax of CDDL is defined in Section 4. Examples of CDDL and related CBOR data instances are defined in Section 5. Section 6 discusses usage of CDDL. A formal definition of CDDL using ABNF grammar is provided in Appendix A. Finally, CBOR keywords are listed in Appendix B.

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. Syntax

4.1. General conventions

The basic syntax is as follows:

- o Each field has a name and a datatype.
- o 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.
- o The datatype in itself MAY be a name of a structure or a map.
- o A name or datatype can consist of any of the characters from the set {'A', ..., 'Z', 'a', ..., 'z', '0', ..., '9', '_'}.
  - * Names and datatypes SHALL NOT start with a numerical character.
  - * Names and datatypes SHALL NOT equal a CDDL keyword, as listed in Appendix B.
  - * Names and datatypes are case sensitive.
  - * Names and datatypes do not appear in the actual CBOR encoding.
  - * It is RECOMMENDED to start a name with a lower case letter, and a datatype with a capital.
- o Comments are preceded by a '#' character and finish with the EOL character.
- o Hexadecimal numbers are preceded by '0x' (without quotes, lower case x), and are case insensitive. Similarly, binary numbers are preceded by '0b'.
- o Strings are enclosed by double quotation '"' characters. They follow the conventions for strings as defined in [RFC7159], section 7.
- o CDDL uses UTF-8 [RFC3629] for its encoding.

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 datatype for other fields. The name is followed by a '{' character and the declarations 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:

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

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

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

4.5. Maps

For maps, CDDL distinguishes between implicit and explicit declarations. Explicit declarations define the datatypes of the keys and values, but not the keys. Implicit declarations define the keys and datatype of associated values. In Implict declarations, the datatypes of the keys can be inferred from the key values.

4.5.1. Explicit Maps

An explicit map declaration is encapsulated in a structure or another map, and has the following form:

```
name: 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 can be replaced by a '.'.

For example, the following could be used as a conversion table converting from an integer or float to a string:

```
*ToString {
  mapper: map( ., tstr );
}
```

4.5.2. Implicit Maps

It is also possible to define a map with predefined keys and type of associated value. The map is defined as a datatype that can be used in structures or maps, but the declaration itself is done outside structure of maps.

The type declaration is as follows:

```
MapDatatypeName: map {
  key1: type1;
  key2: type2;
  ...
}
```

Where MapDatatypeName is the datatype to be used when referring to map, and type1, type2, etc. the datatypes of the value associated with keys key1, key2, etc.

When defining keys in the CDDL, the writing conventions for "value" from [RFC7159], Section 3, are followed. This allows the key datatypes "tstr" and "int".

TBD: float could be signalled in JSON, but how to specify whether it is float(16), float(32) or float(64)? Maybe allow something like "1.3(16)" to indicate a 16-bit float with value 1.3, or "1.7e-2(32)" to indicate a 32-bit float with value 0.017?

The example below the defines a map with display name (as a text string), the name components first name and family name (as a map of text strings), and age information (as an unsigned integer).

```
NameComponents: map {
  "firstName": tstr;
  "familyName" : tstr;
}

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

All key/value pairs are optional from the perspective of CDDL. However, applications MAY enforce mandatory fields as required. Also, it is up to the application how to handle unknown keys, although 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 declaration.

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. Ordering

The declaration of datatypes does not require a specific order. However, it is RECOMMENDED that a datatype that uses another datatype is declared before that other datatype.

For example

```
SmallStructure {
  text: tstr;
  price: float(16);
}

BigStructure {
  innerData: SmallStructure;
}
```

is preferable over

```
BigStructure {
  innerData: SmallStructure;
}

SmallStructure {
  text: tstr;
  price: float(16);
}
```

but both are valid.

Furthermore, it is RECOMMENDED that the CBOR data is encapsulated in an overal structure or map, and all data is encapsulated (at some level) in this overal structure or map.

For example, when defining a message, it would be have an overal structure "Message" that encapsulates the whole message as follows:

```
*Metadata {
  senderName: tstr;
  receiverName: tstr;
}

Message {
  id: bstr;
  data: bstr;
  metadata: Metadata;
}
```

The order of variable instances within structures is fixed by the order of declaration. This means that when a variable A is declared before a variable B, a data instance of A will be encoded in front of a data instance of B.

The ordering of variables in maps is not fixed, as the keys are already an indication for the related value.

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.

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

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

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

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

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.

```
International : map {
  "DE"                 : tstr;             # German
  "EN"                 : tstr;             # English
  "FR"                 : tstr;             # French
  "NL"                 : tstr;             # Dutch
  "ZH-HANS"            : tstr;             # Chinese
}

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

fruitlist              : Fruit*;
```

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
         67 5A 48 2D 48 41 4E 53 # text string length 7, "ZH-HANS"
         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
         67 5A 48 2D 48 41 4E 53 # text string length 7, "ZH-HANS"
         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. Using CDDL

In this section, we discuss several usages for CDDL.

6.1. As a guide to a human user

CDDL can be used to efficiently define the layout of CBOR data, such that a human implementer can easily see how data is supposed to be encoded.

Since CDDL maps parts of the CBOR data to human readable names, editors could be built that use CDDL to provide a human friendly representation of the CBOR data, and allow them to edit such data while remaining compliant to its CDDL definition.

6.2. For automated verification of CBOR data structure

CDDL has been specified such that a machine can handle the CDDL definition and related CBOR data. For example, a machine could use CDDL to verify whether or not CBOR data is compliant to is definition.

The thoroughness of such compliance verification depends on the application. For example, an application may decide not to verify the data structure at all, and use the CDDL definition 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 mechanism that goes as far as verifying that all mandatory map pairs are available.

The matter in how far the data description must be enforced by an application is left to the designers and implementers of that application, keeping in mind related security considerations.

6.3. For data analytics tools

Since CBOR is a data format, it can be expected that more and more data will be stored using the CBOR data format.

Where there is data, there is data analytics and the need to process such data automatically. CDDL can be used for such automated data

processing, allowing tools to verify data, clean it, and extract particular parts of interest from it.

Since CBOR is designed with constrained devices in mind, a likely use of it would be small sensors. An interesting use would thus be automated analytics of sensor data.

7. Open Issues

At least the following issues need further consideration:

- o More extensive security considerations.
- o The key/value pairs in maps have no fixed ordering. However, there may be situations where fixing the ordering may be of use. For example, an decoder could look for values related with integer keys 1, 3 and 7. If the order was fixed and the decoder encounters the key 4 without having encountered key 3, it can conclude that key 3 is not available without doing more complicated bookkeeping.
- o Whether to add signalling of mandatory fields in maps.

8. Change Log

Changes from version 00 to version 01:

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

Changes from version 01 to version 02:

- o Fixed example

Changes from version 02 to version 03:

- o Added information about characters used in names
- o Added text about an overall data structure and order of definition of fields
- o Added text about encoding of keys
- o Added table with keywords
- o Strings and integer writing conventions
- o Added ABNF

Changes from version 03 to version 04:

- o Removed optional fields for non-maps
- o Defined all key/value pairs in maps are considered optional from the CDDL perspective

- o Allow omission of type of keys for maps with only text string and integer keys
- o Changed order of definitions
- o Updated fruit and moves examples
- o Renamed the "Philosophy" section to "Using CDDL", and added more text about CDDL usage
- o Several editorials

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.

Topics that could be considered in a security considerations section that uses CDDL to define CBOR structures include the following:

- o TO DO

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, Joe Hildebrand, Sean Leonard and Jim Schaad.

Appendix A. ABNF grammar

The following is a formal definition of CBOR in Augmented Backus-Naur Form (ABNF, [RFC5234]). We also use the conventions from [RFC5234], Appendix B and [RFC3629], section 4.

```
file             = 1*( structure / map / field)

field            = name ":" type ";" newline

name             = valid-name
```

```
type              = fixed-array / indefinite-array / valid-type

fixed-array       = valid-type "[" 1*DIGIT "]"
indefinite-array = valid-type "*"

structure         = (simple-structure / array-structure) newline
structure-body    = S "{" S 1*field S "}"
simple-structure = name structure-body
array-structure  = "*" name structure-body

map               = name ":" map-header map-body
map-header        = map "(" (valid-type / ".") ","
                    (valid-type / ".") ")"
map-body          = S "{" newline 1*map-entry "}"

map-entry         = map-optional-entry / map-mandatory-entry / comment
map-mandatory-entry = cbor-data ":" type ";"
map-optional-entry = "?" map-mandatory-entry

cbor-data         = cbor-string / cbor-number / cbor-extension
cbor-string       = DQUOTE text DQUOTE
cbor-number       = cbor-leading-numerical / cbor-leading-dot /
                        cbor-hex-number / cbor-binary-number
cbor-leading-dot = '.' 1*DIGIT ['e' 1*DIGIT]
cbor-leading-numerical = 1*DIGIT ['.' 1*DIGIT] ['e' 1*DIGIT]
cbor-hex-number  = hex-prefix 1*HEXDIG
cbor-bin-number  = bin-prefix 1*BIT

hex-prefix        = %d48.120 ; 0x
bin-prefix        = %d48.98 ; 0b
cbor-extension   = *text-char

valid-char        = DIGIT / ALPHA / "_"
valid-name        = 1*valid-char
valid-type        = primitive-datatype / 1*valid-char

primitive-datatype = bool / bstr / float16
                   / float32 / float64 / int
                   / nint / simple / tstr / uint

S                 = *(WS)
newline           = [CR] LF
text-char         = %20-7e
comment-char      = UTF8-char
comment           = "#" *(comment-char) newline
WS                =  SP / HTAB / newline / comment ; white space

; case-sensitive literals
```

```
bool             = %d98.111.111.108
bstr             = %d98.115.116.114
float16          = %d102.108.111.97.116.40.49.54.41 ; float(16)
float32          = %d102.108.111.97.116.40.51.50.41 ; float(32)
float64          = %d102.108.111.97.116.40.54.52.41 ; float(64)
int              = %d105.110.116
map              = %d109.97.112
nint             = %d110.105.110.116
simple           = %d115.105.109.112.108.101
tstr             = %d116.115.116.114
uint             = %d117.105.110.116
```

Appendix B. CBOR keywords

The following table contains an overview of the CDDL keywords.

b64	b64url	bigfloat	bignum
bool	bstr	cbor	decfrac
eb16	eb64	eb64url	epochdt
float	int	map	mime
nbignum	nint	regex	simple
standarddt	tstr	ubignum	uint
uri			

12. Normative References

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

Authors' Addresses

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

Email: bert.greevenbosch@huawei.com

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

Email: sunruinan@huawei.com

Christoph Vigano
University of Bremen

Email: christoph.vigano@uni-bremen.de