Workgroup: Network Working Group
Internet-Draft:
draft-bormann-cbor-cddl-map-like-data-01
Published: 1 June 2021
Intended Status: Informational
Expires: 3 December 2021
Authors: C. Bormann, Ed.
Universität Bremen TZI
B. Moran
Arm Limited
H. Birkholz
Fraunhofer SIT
E. Cormier

# Map-like data in CBOR and CDDL

## Abstract

The Concise Binary Object Representation (CBOR, RFC 8949) 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.

Basic CBOR supports non-ordered maps free of duplicate keys, similar to the way JSON defines JSON objects (RFC 8259). Using the CBOR extension point of tags, tags for a selection of variants of maps and multimaps have been registered, but gaps remain. The present document defines a consolidated set of CBOR tags for map-like data items involving key-value pairs.

The Concise Data Definition Language (CDDL), standardized in RFC 8610, is often used to express CBOR data structure specifications. It provides "control operators" as its main language extension point. The present document defines a number of control operators that enable the description of CBOR data structures that make use of the newly defined tags or that employ the same underlying structures.

This Internet-Draft will expire on 3 December 2021.

## Copyright Notice



## Table of Contents

## 1. Introduction

(See abstract for now.)

### 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.

This specification uses terminology from [RFC8949] and [RFC8610]. In particular, with respect to CDDL control operators, "target" refers to the left hand side operand, and "controller" to the right hand side operand. The terms "array" and "map" (if unadorned) refer to CBOR major type 4 and CBOR major type 5; this is not called out explicitly.

## 2. CBOR tags for map-like data items

This document defines a consolidated set of CBOR tags for map-like entities involving key-value pairs. These tags encode the following meta-data concerning map-like data items:

* the homogeneity of the types of the keys, and of the types of the values;

* whether the order of the key-value pairs carries semantic value ("ordered") or needs to be ignored ("non-ordered");

* the uniqueness of the keys; and

* the major type used to encode the key-value pairs.

Note that the term "ordered" as used in this document is distinct from "sorted" -- "ordered" implies that the order in the data item interchanged conveys a semantically relevant ordering, while a property "sorted" can easily be established after interchange (by, simply, sorting), less often needs to be indicated, and is more complex to indicate as it may need details about the sorting.

### 2.1. Summary

| Tag | LSBs | Homogeneous Value | Homogeneous Key | Ordering | Duplicate Keys Allowed | Data Item | Related Tag |
|---|---|---|---|---|---|---|---|
| 128 | 0000 | No | No | Non-Ordered | No | map | 259 |
| 129 | 0001 | No | No | Non-Ordered | Yes | array | TDB280* |
| 130 | 0010 | No | No | Ordered | No | array | TBD279* |
| 131 | 0011 | No | No | Ordered | Yes | array | |
| 132 | 0100 | No | Yes | Non-Ordered | No | map | 275 |
| 133 | 0101 | No | Yes | Non-Ordered | Yes | array | |
| 134 | 0110 | No | Yes | Ordered | No | array | |
| 135 | 0111 | No | Yes | Ordered | Yes | array | |
| 136 | 1000 | Yes | Yes | Non-Ordered | No | map | |
| 137 | 1001 | Yes | Yes | Non-Ordered | Yes | array | |
| 138 | 1010 | Yes | Yes | Ordered | No | array | |
| 139 | 1011 | Yes | Yes | Ordered | Yes | array | |

Table 1: New CBOR tags defined in this document

*TBD279: https://github.com/Sekenre/cbor-ordered-map-spec/blob/master/CBOR_Ordered_Map.md

*TBD280: https://github.com/ecorm/cbor-tag-multimap

[The intention of the present document is to obviate the need for defining TBD279/TBD280.]

Appendix A.1 provides information about constructs in a few programming languages that are related to the tags being defined.

### 2.2. Key/Value Type

Bits 2 and 3 of the tag provide information on the map's key and value types:

*0b00xx Unspecified: There is no specified type for the map's keys and values

*0b01xx Homogeneous Key: All keys have the same data type

*0b10xx Homogeneous Key/Value: All values have the same data type in addition to all keys having the same data type (the types for keys and values may be distinct).

The semantics for homogeneity shall be the same as for [RFC8746] homogeneous arrays (tag 41). That is, "which CBOR data items constitute elements of the same application type is specific to the application" (Section 3.2 of [RFC8746]).

Maps with arbitrary keys and homogeneous values are considered unusual, so they are left out of this specification so that fewer tag numbers need to be allocated (12 instead of 16).

### 2.3. Ordering

Bit 1 of the tag represents the map's ordering semantics:

- 0: The order of key-value pairs is unspecified
- 1: Key-value pairs are encoded in the same order in which they were inserted

### 2.4. Key Uniqueness

Bit 0 of the tag represents the uniqueness of the map's keys.

- 0: Keys are unique within the map
- 1: Keys may be duplicated (i.e., multimaps)

### 2.5. Data Item

All these map-like data items could be represented as a tag with an enclosed array of alternating key-value pairs, as in:

```
129(["key1", 1, "key2", 2])
```

However, representing the key-value pairs as a CBOR map for those cases where this is possible enables generic decoders that are oblivious of these tags to represent the data in a more appropriate platform type.

Specifically, the key-value pairs are represented as a map if and only if

- the ordering is unspecified and
- the keys are unique;

otherwise, they are represented as an array of alternating keys and values ("flattened alist", see Figure 1).

```
FAList<K, V> = [* (K, V)]
```

Figure 1: CDDL for order-preserving representation of maps

**Issue**: [MAPREP] discusses alternative representations of (ordered and other) maps. How much of this do we need to address here?

### 2.6. Related Tags (Informative)

#### 2.6.1. Tag 259

Specification: https://github.com/shanewholloway/js-cbor-codec/blob/master/docs/CBOR-259-spec--explicit-maps.md

The above defined tag 128 may be used instead to guide a JavaScript decoder into interpreting a CBOR map as a JavaScript Map instead of an Object.

#### 2.6.2. Tag 275

Specification: https://github.com/ecorm/cbor-tag-text-key-map

The above defined tag 132 may be used instead to guide a decoder into interpreting a CBOR map as a JavaScript-like Object having only text string keys. The decoder would have to verify the first key to establish that the map has homogeneous text string keys.

#### 2.6.3. Tag TBD279

Draft specification: https://github.com/Sekenre/cbor-ordered-map-spec/blob/master/CBOR_Ordered_Map.md

The above defined tag 130 may be used instead to encode map-like data items where the order of the key-value pairs is semantically significant.

#### 2.6.4. Tag TDB280

Draft specification: https://github.com/ecorm/cbor-tag-multimap

The above defined tag 129 may be used instead to encode a multimap as an array of key-value pairs.

## 3. CDDL Support for Map-Like Data Items

The Concise Data Definition Language (CDDL), standardized in RFC 8610, provides "control operators" as its main language extension point.

The present document defines a number of control operators that enable the use of group notation (enclosed in a CDDL map) to specify any of the above map-like data structures:

| Name | Purpose |
|---|---|
| .omm | Ordered (Multi-)Map |
| .nomm | Non-Ordered (Multi-)Map |
| .unique | Uniqueness requirement |

Table 2: New control operators in this document

### 3.1. Map notation for map-like data items

[needs better examples]

CDDL already can describe both arrays of alternating keys and values and maps (non-ordered and with unique keys). The two control operators .omm and .nomm introduced in this section enable the use of CDDL map notation for map-like types beyond actual maps, increasing readability and possibly even reusability.

In a simple example that provides an non-ordered collection of zero or more home addresses and zero or more work addresses, each labeled as such, we use traditional map notation to describe that collection:

```
[* (text, any)] .nomm {
  * home: address
  * work: address
  $$more-addresses
}
```

The .omm and .nomm control operators convert a group definition enclosed into a CDDL map given as a controller type into an array type given as the target type. The controller type given is unwrapped (Section 3.7 of [RFC8610]) into a group. Keys and values of the entries in that group are then alternatingly matched as elements in the target array. Note that both target and controller type can contribute to the shaping of the data; declaring the key type as text limits what can be added to the $$more-addresses socket.

.omm and .nomm differ in the semantics of the array type created: .omm defines an ordered (multi)map, i.e., the order of the key/value element pairs in the array matters, while .nomm defines an

non-ordered (multi)map, i.e., data items that present the same set of key/value pairs in different orders are equivalent.

The ability to specify specific ("homogeneous") types is provided by the ability to specify the target type, as in the example above.

Note that there is not strictly a need to define a control operator for building non-ordered maps with non-duplicate keys, as existing CBOR maps already fill this role, however the use of a map type as the target is allowed for symmetry (implying uniqueness of the keys), allowing the following:

```
{* text => any} .nomm {
  ? home: address
  ? work: address
  $$more-addresses
}
```

### 3.2. Uniqueness

The .unique control annotates the target as requiring uniqueness, within the enclosing container(*), of its value, among the other data items in that enclosing container that are also marked .unique, under the same label (given as the controller).

E.g.,

```
feature-set = [* feature .unique "set"]
ordered-pairs-with-unique-keys-and-values =
  [* (any .unique "key", any .unique "value") ]
```

defines a feature-set as an array of zero or more feature values that need to be all different (as they are unique under the label set), and ordered-map-with-unique-keys-and-values as an array of zero pairs of keys and values, where the keys need to be unique among themselves and the values need to be unique among themselves (the latter example could employ an .omm or .nomm operator to further restrict what can be in these keys and values).

> Discussion: (*) while it is probably not a big problem to define what exactly the "enclosing" container is, it may be useful to actually define a larger scope of the uniqueness. CDDL currently does not have a way to establish and point to such a larger scope; we might define one ad hoc here or leave that for later extension.

## 4. CDDL typenames

For the use with CDDL [RFC8610], the typenames defined in Figure 2 are recommended unless there is a need for more specific shaping of the data.

```
anymap = {* any => any}

tbd128 = #6.128(anymap)
tbd129 = #6.129([* (any, any)] .nomm anymap)
tbd130 = #6.130([* ((any .unique "mm"), any)] .omm anymap)
tbd131 = #6.131([* (any, any)] .omm anymap)
tbd132<k> = #6.132({* k => any})
tbd133<k> = #6.133([* (k, any)] .nomm anymap)
tbd134<k> = #6.134([* ((k .unique "mm"), any)] .omm anymap)
tbd135<k> = #6.135([* (k, any)] .omm anymap)
tbd136<k,v> = #6.136({* k => v})
tbd137<k,v> = #6.137([* (k, v)] .nomm anymap)
tbd139<k,v> = #6.138([* ((k .unique "mm"), v)] .omm anymap)
tbd139<k,v> = #6.139([* (k, v)] .omm anymap)
```

Figure 2: Recommended typenames for CDDL

**Issue**: fill in better names for tbdnnn

Note that there is no need to call out the uniqueness of the keys explicitly in tbd128, tbd132, or tbd136, as the use of maps as a representation format already provides that key uniqueness.

## 5. IANA Considerations

### 5.1. Tags

IANA is requested to allocate the tags of Table 1 in the CBOR tags registry [IANA.cbor-tags], using this document as the specification reference.

The allocations are requested to be assigned from the "specification required" space (24..255). The values in the column labeled "Tag" in Table 1 are suggested as the allocated tag numbers.

### 5.2. CDDL control operators

This document requests IANA to register the contents of Table 3 into the CDDL Control Operators registry [IANA.cddl]:

| Name | Reference |
|---|---|
| .omm | [RFCthis] |

| Name | Reference |
|---|---|
| .nomm | [RFCthis] |
| .unique | [RFCthis] |

Table 3: New control operators to be registered

## 6. Implementation Status

TBD

## 7. Security considerations

The security considerations of [RFC8610] apply.

## 8. References

### 8.1. Normative References


**[IANA.cbor-tags]** IANA, "Concise Binary Object Representation (CBOR) Tags", <http://www.iana.org/assignments/cbor-tags>.

**[IANA.cddl]** IANA, "Concise Data Definition Language (CDDL)", <http://www.iana.org/assignments/cddl>.

**[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>.

**[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>.

**[RFC8746]** Bormann, C., Ed., "Concise Binary Object Representation (CBOR) Tags for Typed Arrays", RFC 8746, DOI 10.17487/RFC8746, February 2020, <https://www.rfc-editor.org/info/rfc8746>.

**[RFC8949]** Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

### 8.2. Informative References

**[MAPREP]** Bormann, C., "Re: [Cbor] "ordered hash"", cbor@ietf.org mailing list message, 30 July 2020, <https://mailarchive.ietf.org/arch/msg/cbor/5MuDSyPivZ7JfPhsfwCaW2usFHQ>.

## Appendix A. Implementation Considerations

This non-normative appendix provides information about the use on implementations of the tags and control operators defined.

### A.1. Programming Language Containers (Informative)

The following subsections describe how the tags in this document relate to various programming language containers. Containers that are not part of the programming language or its standard libraries are not considered here.

The *Encoding Tag* column in the following tables provide the recommended tag that best represents the given container type. For example, it's possible to use tag 132 for encoding an ECMAScript Map if all keys happen to be of the same type, however tag 128 is more general and applies to any Map. When encoding an ECMAScript Object, tag 128 would be technically correct but is too general; tag 132 best presents the fact that an Object has text keys only.

The *Decodable Tags* column in the following tables, are for data items can be decoded into the destination container without having to inspect the following:

- the uniqueness of the keys,
- the ordering of the keys, and,
- the data types of **every** keys/value pair.

It may however be necessary to inspect the data types of the **first** key-value pair in the case of tags representing homogeneous keys/values.

#### A.1.1. ECMAScript

| Container | Encoding Tag | Decodable Tags |
|---|---|---|
| Object | 132 | 132, 136 |
| Map | 128 | 128, 132, 136 |
| Array of pairs | 131 | All |

Table 4

### A.1.2. Python

| Container | Encoding Tag | Decodable Tags |
|---|---|---|
| TypedDict | 136 | 136 |
| namedtuple | 132 | 132, 136 |
| dict | 128 | 128, 132, 136 |
| OrderedDict | 130 | 130, 134, 138 |
| list of 2-tuples | 131 | All |

Table 5

### A.1.3. C++

| Container(s) | Encoding Tag | Decodable Tags |
|---|---|---|
| Map<K, T> | 136 | 136 |
| Map<K, D> | 132 | 132, 136 |
| Map<D, D> | 128 | 128, 132, 136 |
| MultiMap<K, T> | 137 | 137 |
| MultiMap<K, D> | 133 | 133 |
| MultiMap<D, D> | 129 | 128, 129 |
| Sequence<Pair<K, T>> | 139 | [136, 139] |
| Sequence<Pair<K, D>> | 135 | [132, 139] |
| Sequence<Pair<D, D>> | 131 | All |

Table 6

Legend:

*K: Static key type

*T: Static value type

*D: Suitable dynamic type, such as std::any or std::variant

*Map: std::map or std::unordered_map

*MultiMap: std::multimap or std::unordered_multimap

*Sequence: Sequence container that maintains order (e.g. std::vector)

*Pair: Object containing a key and a value, such as std::pair, or std::tuple.

Note that a C++ std::map stores its key-value pairs in a sorted fashion, and does **not** preserve insertion order in the same manner as Python's OrderedDict.

## A.2. CDDL Implementation Considerations

TBD

## Acknowledgements

The CBOR tags defined in this document were developed by Emile Cormier under the sponsorship of Duc Luong, based on discussions with Kio Smallwood and Joe Hildebrand. The CDDL control operators defined in this document were developed by Carsten Bormann, Brendan Moran, and Henk Birkholz.

## Contributors

Kio Smallwood

Joe Hildebrand

## Authors' Addresses

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

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

Brendan Moran
Arm Limited

Email: Brendan.Moran@arm.com

Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

Emile Cormier

Email: emile.cormier.jr@gmail.com