Concise Binary Object Representation Maintenance and Extensions R. Mahy
Internet-Draft   17 October 2025
Intended status: Informational
Expires: 20 April 2026

CBOR Pointer: Selecting Elements of Concise Binary Object Representation (CBOR) Documents
draft-mahy-cbor-pointer-00

Abstract

CBOR Pointer is a syntax to identify a single CBOR value from a CBOR document with an arbitrarily complex nested structure. It is analogous to JSON Pointer.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://rohanmahy.github.io/cbor-pointer/draft-mahy-cbor-pointer.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mahy-cbor-pointer/.

Discussion of this document takes place on the Concise Binary Object Representation Maintenance and Extensions Working Group mailing list (mailto:cbor@ietf.org), which is archived at https://www.ietf.org/mail-archive/web/cbor/current/maillist.html. Subscribe at https://www.ietf.org/mailman/listinfo/cbor/.

Source for this draft and an issue tracker can be found at https://github.com/rohanmahy/cbor-pointer.

Status of This Memo

This Internet-Draft will expire on 20 April 2026.

Table of Contents

1. Introduction

CBOR Pointer is a syntax for identifying a single arbitrary subtree or element of a CBOR [RFC8494] Document or a CBOR sequence. It provides functionality analogous to JSON Pointer [RFC6901] but supporting the full range of CBOR types.

2. Conventions and Definitions

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.

3. Definition

A CBOR Pointer is an array consisting of pathspecs. The entire array can be implicitly typed or explicitly typed. The first pathspec operates on the root of the CBOR or CBOR sequence document as the parent element. The entire CBOR document matches the CBOR Pointer [].

Evaluating a CBOR Pointer returns either an array containing a single valid CBOR element, or returns null.

3.1. Implicit Pathspecs

The semantics of an implicit pathspec depend on the type of the parent element.

* If the parent element is an array, it returns the appropriate element:
  - if the pathspec is an unsigned integer, it matches the element at that zero-based position from the start of the array;
  - if the pathspec is a CBOR negative integer, hexadecimal 0x20 matches the last element of the array, with higher numbers moving backwards through the array
* If the parent element is a map, the pathspec matches if it matches one of the map keys of the map. It returns the value of the map key.
* If the parent element is a tag, the pathspec matches if it matches the tag number. It returns the value inside the tag.
* If the parent element is a byte string, the parent element is re-evaluated as embedded CBOR. The pathspec is evaluated as above if the type after byte string decoding is an array, map, or tag.
* If the root element is a CBOR sequence, and the pathspec is evaluated as if the entire sequence were wrapped in an array.

3.2. Examples with Implicit Pathspecs

Given the following source document, the table below gives the corresponding result.

```
777([
  [
    [1, "two", 3],
    [4, "five", 6]
  ],
  {
    1: "abc",
    -18: h'1234',
    "x": null,
    35: 1(1760686166),
    "y": [ "l", "m"]
  },
  <<{
    2: 45,
    "pdq": false
  }>>,
  27,
  h'abcdef'
])
```

CBOR Pointer	Result
[77]	null
[777, 3]	[27]
[777, 9]	null
[777, null]	null
[777, 0]	[[[1,"two",3],[4,"five",6]]]
[777, 0, 1]	[[4, "five",6]]
[777, 0, 1, 1]	["five"]
[777, 1, 1]	["abc"]
[777, 1, -18]	[h'1234']
[777, 1, -18, 1]	null
[777, 1, "x"]	[null]
[777, 1, 35]	[1(1760686166)]
[777, 1, 35, 1]	[1760686166]
[777, 1, "y"]	[["l","m"]]
[777, 1, "y", 1]	["m"]
[777, 1, "z"]	null
[777, 2]	[h'49a202182d63706471f4']
[777, 2, 2]	[45]
[777, 2, "pdq"]	[false]
[777, 2, 0]	null

Table 1

3.3. Explicit Pathspecs

Explicit CBOR Pointers use tags to match a specific type of element for each pathspec. If the type of the parent element matches the expected type, the matching rules and return values are the same as for implicit pathspecs, except that in explicit pathspecs, byte string encoded strings are unwrapped in a separate pathspec. Explicit CBOR Pointers are always wrapped in the tag <TBD1>. Each element in an explicit CBOR Pointer is either the simple value <TBD0> for byte string encoded elements, or a pathspec tagged with one of the following tags:

Data Type	Tag
array	TBD2
map	TBD3
tag	TBD4
sequence	TBD5

Table 2

Converting one of our implicit pathspec examples ([777, 1, "y", 1]) into explicit pathspecs, gives us:

```
TBD1([          # Explicit pathspecs
    TBD4(777),  # Tag 777
    TBD2(1),    # 2nd Array element
    TBD3("y"),  # Map key "y"
    TBD(1)      # 2nd Array element
])
```

Both the implicit and explicit version return the value ["m"]. However, an explicit pathspec tag referring to a different type would return null. Consequently explicit pathspecs are useful where different types could be in the same location and the distinction is semantically meaningful.

Explicit pathspecs involving embedded byte strings require an additional pathspec element. For example, the equivalent of the implicit pointer [777, 2, 2] (which returns [45]) is the following:

```
TBD1([            # Explicit Pathspecs
    TBD4(777),    # Tag 777
    TBD2(2),      # 3rd Array element
    simple(TBD1), # decode byte string
    TBD3(2)       # Map key 2
])
```

This property of explicit pathspecs makes it possible to return the entire decoded value of an encoded byte string. For example, the following explicit pointer applied to our original example:

```
TBD1([            # Explicit Pathspecs
    TBD4(777),    # Tag 777
    TBD2(2),      # 3rd Array element
    simple(TBD1)  # decode byte string
])
```

returns [{2:45, "pdq":false}] as its result.

3.4. Array Filters

Array filters allow selecting a single array element by evaluating the contents of those elements against another CBOR Pointer. This filter pointer is evaluated against each array element inside the parent element where the filter was invoked, in turn. An array filter is tagged with tag TBD6.

For example, if a filter were invoked at the following parent element of our initial example document, the filter pointer is evaluated once for each of the two elements of the parent.

```
[
  [1, "two", 3],
  [4, "five", 6]
]
```

The CBOR Pointer Filter below selects the entire matching array element under the parent element, where the second element ([TBD2(1)]) matches the value "five":

```
TBD6([           # Array Filter
    [            # Per-element CBOR Pointer
        TBD2(1)  # 2nd Array element
    ],
    ["five"]     # value to match per-element pointer result
])
```

If multiple elements or no elements would be returned after evaluating an array filter, the result of the entire array filter is null.

When evaluating the CBOR Pointer (containing an array filter) below, against the original example, the result is [6]:

```
TBD1([            # Explicit Pathspecs
    TBD4(777),    # Tag 777
    TBD2(0),      # 1st Array element
    TBD6([        # Array Filter
      [             # Per-element CBOR Pointer
        TBD2(1)       # 2nd Array element
      ],
      ["five"]      # value to match per-element pointer result
    ]),
    TBD2(2)       # 3rd Array element
])
```

4. Security Considerations

TODO Security

5. IANA Considerations

TO DO register 6 tags (TBD1 through TBD6) and 1 simple value (TBD0).

6. References

6.1. Normative References

[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/rfc/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/rfc/rfc8174>.

[RFC8494] Wilson, D. and A. Melnikov, Ed., "Multicast Email (MULE) over Allied Communications Publication (ACP) 142", RFC 8494, DOI 10.17487/RFC8494, November 2018, <https://www.rfc-editor.org/rfc/rfc8494>.

6.2. Informative References

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/rfc/rfc6901>.

Acknowledgments

TODO acknowledge.

Author's Address

Rohan Mahy
Email: rohan.ietf@gmail.com