CBOR EDN readability, etc.

Rohan Mahy

10-Jun-2026 Interim Meeting

Reminder: What is EDN for?

Ideation only

- Whiteboards
- Example snippets
- Informal examples

- Don’t need to conform to a syntax as long as example is understood

EDN to CBOR

- Test vectors
- CI / Configuration files
- Automatically generated documents
- Formal examples in specifications

Implementers need to code defensively in these cases

Implementation had better support **all** the extensions in the EDN text

CBOR to EDN

- Take a CBOR sequence on the wire or in a file on disk: make it easy to visualize. **Diagnostics**!

Generated EDN does not need all the bells and whistles to be useful

Better be able to **express all valid CBOR** and not hide unusual map order, non-preferred encoding, indefinite length, non-trivial NaNs.

Otherwise, little concern about which extensions are used

Guidance about default style might be welcome

Why is Readability Important?

- In EDN to CBOR case
  - make it easier to spot genuine errors, and attacks on EDN source files that can become supply chain / CI attacks or similar.
- In CBOR to EDN case
  - make sure unusual CBOR is noticed in diagnostics.
  - there are several uses cases in CBOR where there is a hidden channel that can be used for exfiltration:
    - ex: private information about a user in web tokens
    - ex: secret key / private key information in crypto
- In General: adoption
  - If developers understand an EDN document in a spec or example they are more likely to use CBOR

Readability Topics

- Backtick-quoted strings
- Comments
- Optional Commas (and Extra Trailing Comma)
- String Concatenation
- Indefinite Length Strings (streamstring)
- Encoding Indicators
- Defaults (covered dead last)

Backtick-quoted strings

- Humans can’t easily distinguish more than a handful of backticks in a row.

  ```````` ⬅ Quick, how many backticks was that?

- Allow leading and trailing backticks is hard for humans to see/read/understand
- ```a = ``foo`````  ```a = ``foo``` + “````”
- ````````````
  ``foo`` = ``bar``````````````
- And you can also use concatenation instead
- But my editor will color them for me so I know what is in the string! Nope. For many editors, not with the syntax we have now.
- Alternatives
  - Drop backtick-quoted strings — JSON lived with them for decades, we lived with it in EDN for 10 years.
  - Restrict us to maximum 8 backticks ; don’t allow leading or trailing backticks
  - Use GitHub “fenced” backticks (but it makes some leading space not possible to represent)
    - ``` a = ``foo`` ```
  ````````

Pathological Comments

- We now have 4 types of comments in EDN: `/ /`, `/* */`, #, and `//`

```
`` / `` / `` / `` / ``

`` /* `` /* `` */ `` */ ``

/ * "**" /'*'// + "**" */ + '*'
```

- Unusually, we allow comments in quoted strings: for backwards compatibility with `h''` and `b64''` (before we had concatenation). b64 only allows #.

```
h'/this is a comment\/suggestion/ 01234567'
```

- Option: Undo our recent addition of C-style comments.
- Suggestion 1: Don't allow EDN comments in ***new*** quoted string extensions. If the syntax is complicated enough to require comments, it probably has ***another*** native comment format.
- Suggestion 2: Style guide: only use / / comments before map key names and after values

Optional Commas

- Optional Commas: between lists of values require a comma OR space OR a comment

```
{_1 1_1:-7 16:295}

{/alg/1:-7/typ/16:295}

[1/iss/2/sub/4/exp/8/cnf/]

1/ /2/ /3/4/5//6/7/8
```

- Result: readable when replaced with a newline, but other usually not very clear.
- Suggestion: maybe the cure was worse than the disease?
- Suggestion: style guide: always use comma separators for clarity when writing EDN
- What about Optional Extra Commas at end of the list: They are “Harmless to Humans.”

String Concatenation

- Currently we can concatenate strings with an explicit + . (Briefly we concatenated using whitespace. That was wickedly dangerous)

```
‘Please get the following:\n’ +
‘- bread\n’
‘- milk\n’,

h’e2d46ccf013d0c7728d89d97e659442396e71fb50ec477e9e16248a9ca344949’ +
h’4d796a9ee96d128b7821803e7621a9deca8ee4b0746d39b2ae5a1de0b4731610’,

b64’kMnsizEB5tylDpmevhzmzJ1U9PWFmqaJBgp_JarKOJ2E’ +
b64’dqWcdN595RWlJeZxIqASh6Blhod_2ZMmSvQ-s0IFW68’,
```

Concatenated string fragments is perhaps the only place in EDN where encoding indicators are nonsensical

- But it is not always obvious the resulting type (text string or byte string), especially if one of the string fragments is an app-extension

```
“” + h’00’ + “hello” + h’09’ + “world”

‘’ + ip’192.0.0.2’ + h’18’ + hash<<‘en0/’ + ip’192.0.0.2’>>

h’123456’ + ...
```

- Proposal from Vadim (has some nice properties but perhaps harder to read):

```
tstr<<h’00’, “hello”, h’09’, “world”>>

bstr<<ip’192.0.0.2’, h’18’, hash<< bstr<<‘en0/’, ip’192.0.0.2’>> >> >>
```

Indefinite-length strings (streamstring)

- Again, strings in CBOR are super complicated. We have another syntax which needs to figure out the type of string it is working with.

```
(_ ""+h'09', "hello", h'20'_3, "world" )
```

```
(_ h'00'_1, '' + `don't panic`, ip'192.0.0.2', hash<<'foo'>>)
```

- Vadim's proposal is pretty cool, but requires extensions to return CBOR bytes

```
tstr-i<< h'09', "hello", h'20', "world" >>
```

```
bstr-i(_ h'00'_1, `don't panic`_0, ip'192.0.0.2', hash<<'foo'>>)
```

(it also allows an encoding indicator here ^^)

and we could get rid of `''_` and `""_`

Encoding Indicators

- Encoding indicators are an area that seems to be poorly implemented overall.
- They are very important for test vectors in the EDN to CBOR case, and in the CBOR to EDN diagnostic use case if they aren't in preferred form.
- There is no defined way to express the encoding of a concatenated string.
- Do we really need a registry? If there were a change to CBOR to add u128s for example, we could add an _4 encoding indicator then. We don't need a registry for that.
- Informal proposal from Carsten was that we might use encoding indicators to signal determinism, for example.
  - I think Carsten's proposal makes the extensibility model really messy
  - What if app extensions controlled the encoding their own content?

mt	examples
0	`1_1, 0x4711_3`
1	`-1_1`
2	`'A'_1`
3	`"A"_1`
4	`[_1 "bar"]`
5	`{_1 "bar": 1}`
6	`1_1(4711)`
7	`1.5_2, 0x4711p+03_3`

Table 1: Examples of Encoding Indicators for Different Data Items (mt = major type)

Encoding Indicator	Description	Reference
_	Indefinite Length Encoding (ai=31)	RFC8949, RFC-XXXX
_i	ai=0 to ai=23	RFC-XXXX
_0	ai=24	RFC8949, RFC-XXXX
_1	ai=25	RFC8949, RFC-XXXX
_2	ai=26	RFC8949, RFC-XXXX
_3	ai=27	RFC8949, RFC-XXXX
_4	Reserved (for ai=28)	RFC-XXXX
_5	Reserved (for ai=29)	RFC-XXXX
_6	Reserved (for ai=30)	RFC-XXXX
_7	Reserved (see _)	RFC8949, RFC-XXXX

Table 9: Initial Content of Encoding Indicator Registry

Non-Readability Topics

- What do Application Extensions “return”?
- Stand-in syntax
- Metadata we put at the start of an EDN text (like a pragma?)

What do Application Extensions “return”?

- This is actually pretty ambiguous in the spec. I think there are two mental models:
- Carsten: the “output” of an extension is a legal EDN text.
  - `ip’192.0.0.2’ ➡ h’C0000002’`
  - `IP’192.0.2.0/24’ ➡ 52([24, h’C0000200’])`
- Joe, Vadim: the output of an extension is a type and CBOR bytes:
  - `ip’192.0.0.2’ ➡ bytes, C0000002`
  - `IP’192.0.2.0/24’ ➡ tag 52, 82 1818 44C0000002`
  - Allows Vadim’s `tstr-i<< >>` and my `sort-map<< >>` extensions.

Stand-in syntax: (considered harmful)

- The idea behind stand-in encoding is that it is useful to have a CBOR document that does not reflect the original intent of the EDN, but could be later processed into a CBOR document with the actual intent.
- I am not aware of anyone who has deployed such a system. So who asked for this?
- Meanwhile in our **actual use cases** for EDN to CBOR conversion (test vectors, configuration, CI) this is exactly the opposite of what we want. It is a huge potential attack vector.
- Suggestion A: remove it
- Suggestion B: make it configurable and off by default. But how would you signal it? …. hold my beer please…

Metadata in EDN

Metadata we put at the start of an EDN text (like a pragma?)

EDN to CBOR example

```
/*!
required: [cri, hash, e]
standin: disable
ellipsis: fail
map-sort: deterministic
encoding-indicators: allowed
indefinite-encoding: forbidden
!*/
```

CBOR to EDN example

```
/*!
required: []
text-escaping: all-non-ascii
ellipsis: long-bstr
allow-encoding-indicators: true
!*/
```

Defaults / Style

- pretty print like JSON defaults
- comments
  - don’t use multi-line comments unless the line only consists of comments
  - use any comment type after a value before the end of the line
  - only use # comments in h” and b64”
  - only time you should have comment before a value on the same line is a / / comment before an integer map key
- in diagnostic mode (CBOR to EDN)
  - render maps as they appear on the wire. render indefinite encoding; add encoding indicators for non-preferred encoding.
  - use h” unless byte string is all printable ascii or Unicode Letter category characters in the same language family.
  - If CDDL data model is available, comment map keys and other integer values. Decode embedded bstrs.
  - Can be configured to represent long byte strings as an h” with the first 6 octets, an ellipsis, and the last 6 octets