CBOR@IETF123 2025-07-25

Agenda today

CBOR	tags/apps	EDN	CDDL
		edn-literal	
CDE			
	packed		

Objectives for this meeting: EDN

- EDN-literal
  - Decide raw-string proposal (20250709 interim)
    - Implemented in edn-abnf;
      To do: Prepare PR for EDN spec
  - Conduct objectives discussion
  - Clear way to WGLC

Objectives for this meeting: CDE

- CDE
  - Complete NaN payload narrowing/widening discussion
    - Stick with RFC 8949 decisions; merge (PR #37)
  - Establish focus of CDE BCP
    - Do we need more text that is unrelated to CDE?
  - Clear way to progress — another WGLC?
    Compare -08 to -12

Objectives for this meeting: Packed CBOR

- Packed
  - Decide A=12/B=8/C=8
  - See Martine's DNS results, hackathon results
  - Prepare some final interop work
  - ➔ WGLC ~ end of August

Objectives for this meeting: EDN

- EDN-literal
  - Decide raw-string proposal (20250709 interim)
    - Implemented in edn-abnf;
      To do: Prepare PR for EDN spec
  - Conduct objectives discussion
  - Clear way to WGLC

Raw Strings

- Escaping hell: backslashes double with every level
  `re'[\\-\\w\\d]+'`
- Raw strings: Don't escape on backslashes
  `re`[\-\w\d]+``
- Minimizes processing of backslash-heavy app-extensions
  - technically and mentally

Raw Strings: Small Print

— One leading newline is swallowed

```
text text text
```

— Swallowing: also workaround for starting with backtick

```
re``
`foo'``
```

— Ending with backtick: Additional backticks

```
re``
`foo```
```

Raw String: ABNF

```
rawstring       = startrawdelim
                  [newline] ; swallow up to one leading newline
                  rawcontent
                  matchrawdelim
rawdelim        = 1*"`"
startrawdelim   = rawdelim ; remember length
matchrawdelim   = <rawdelim: only if not shorter than length>
rawchars        = 1*(%x09 / %x0a / %x0d / %x20-5f / %x61-7e / NONASCII)
rawcontent      = 1*(rawchars/shortrawdelim)
shortrawdelim   = <rawdelim: only if shorter than length>
```

Raw String: Expanding the prose-val (e.g., abnftt)

```
rawstring       = startrawdelim
                  [newline] ; swallow up to one leading newline
                  rawcontent
                  matchrawdelim
                  {def ast; rawcontent.text_value.gsub("\r", "") + (
                       tv = matchrawdelim.text_value
                       dv = startrawdelim.text_value.length
                       tv[dv..]) end}
rawdelim        = 1*"`"
startrawdelim   = rawdelim&{|(rd)|@rawdelimlength = rd.text_value.length}
shortrawdelim   = rawdelim&{|(rd)|rd.text_value.length < @rawdelimlength}
matchrawdelim   = rawdelim&{|(rd)|rd.text_value.length >= @rawdelimlength}
rawchars        = 1*(%x09 / %x0a / %x0d / %x20-5f / %x61-7e / NONASCII)
rawcontent      = 1*(rawchars/shortrawdelim)
```

Raw Strings: redux

- \+ Significant usability gain with backslash-heavy strings
- \+ Reasonably easy to implement
  - PEG implementations: Semantic predicates
  - Traditional parser generator: can use a "scanner/parser" setup
- – uses ABNF "prose-val" (<...>)

```
matchrawdelim  = <rawdelim: only if not shorter than length>
shortrawdelim  = <rawdelim: only if shorter than length>
```

EDN: Objectives

EDN draft: guided by objectives

— Some tacit

— Some more or less relevant depending on perspective

(In part based on input from Rohan Mahy. Thanks!)
See also Section 1.3 of edn-literals.

EDN Objectives (tacit):

Technical excellence

— Design for decades
(Outcome-oriented design)

— Heed Protocol Design 101

EDN Objectives:

Invariants

1. Valid JSON is valid EDN
2. EDN can represent any valid CBOR
3. EDN can represent any valid CBOR
   while using a limited source character repertoire

— More precision:

1. For every CBOR encoded data item, there should be some EDN [of some given quality standards] that converts into that encoded CBOR data item.
2. For every CBOR encoded data item, there should be some EDN [of some given quality standards] that converts into an encoded CBOR data item equivalent to the given CBOR encoded data item.

Variation 2:
• allows some limitations in encoding indicators, and
• in particular makes generator/converter implementations valid that do not implement (all) encoding indicators at all.

(Note that even Variation 2 still is not fully fulfilled, as there is no EDN to represent NaNs with non-zero payloads (including signalling NaNs). This can be addressed using appropriate app-extensions.)

EDN Objectives:

Usability desiderata: Humane

(Not really operationalize beyond "I know it when I see it", but very important objectives nonetheless:)

— EDN is human readable and human writable

Different levels of focus:

- A human reading the EDN can figure out what what the corresponding CBOR looks like
  - Some EDN is designed to make the connection to the encoded data item very obvious
  - Some EDN is designed to stay closer to the application
    (Might be used together with pretty-printed hex.)

EDN Objectives:

Repertoire issues

— An EDN generator can be parameterized to use only those unicode characters inside a target repertoire; any characters outside the target repertoire in strings can be represented in a familiar and convenient manner

  — Note: already true of JSON and, with invariant 1, constrains the minimum support that can be provided.

- An EDN file reliably retains its semantics when moved between Windows and Linux (line separators!)
- Can represent binary data in hex or base64(url) with optional whitespace (including new lines) and optional comments
- When quoting text strings and binary strings, it is not necessary to have a lot of escaping
  - Better support ➔ raw strings proposal

EDN Objectives:

Implementability

- Semantics: EDN builds from and in terms of CBOR components
- Syntax (parser focus):
  We can represent the EDN grammar in a specification.
  - Practically ➔ use of ABNF for the surface syntax.
    - ABNF = context-free grammar, limits expressibility.
    - raw-string syntax is easily implementable via modern parser generators (possibly requiring a "scanner" layer for some of them), but requires the use of (non-formalized) prose-val elements in the ABNF grammar.

- Parsing is not ambiguous
  - The term "ambiguous" is ambiguous when it comes to parsing…
  - Alternative formulation:
    The parsing cannot lead to two valid interpretations of an EDN input that do not lead to equivalent CBOR
- Parsing unlikely to cause overflows etc.
  - The EDN language does not provide any artificial limitations that could be used to provide a hard limitation on stack use (or equivalent).
  - Implementations need a way to control resource usage due to unlimited file size or unlimited nesting

- Prevention against confusion attacks
  - This appears to have a humane aspect to it, and I sure wish there were a way to reliably prevent human confusion.

EDN Objectives:

Extensibility

CBOR ecosystem == extensibility, e.g.:

- CBOR: Tags
- CDDL: Control Operators
- EDN: App-extensions

From mere extensibility to an "extension point"

EDN Objectives:

Extension point app-extension

Extension point addresses:

— base spec designer (we right now)
— base spec implementer (some of us)
— extension spec designer (not always we)
— extension implementer
— extension user

EDN Objectives:

Framing extensibility into an Extension point

Extension point:
Clear framework for an extension

- Isolation:
  - Cannot break main mechanism with an extension
  - Extension designer/developer only need to think about their extension

➔ Litmus test: Can I implement an App-extension in TCL?

Enabling a pluggable architecture

EDN Objectives:

...building from CBOR components...

simplifying generalization:
Feed any CBOR (sequence) into an app-extension
not just string: `foo'bar'`, but also `foo<<'bar', 4711, true>>`

Example app-extension that interacts with CDDL:
draft-bormann-cbor-edn-mapkey (Christian Amsüss' idea)
Look up integer map labels for map keys in CDDL

(Compare JSON usage in ISO mDoc example appendices)

```
mapkey<<"Claims-Set",
  {
    "sub": "42-50-31-FF-EF-37-32-39"
    "cnf": {
      "COSE_Key": {
        "kty": "OKP"
        "kid": h'00'
        "crv": "X25519"
        "x": h'b1a3e89460e88d3a8d54211dc95f0b90
              3ff205eb71912d6db8f4af980d2db83a'
      }
    }
  }
>>
```

Tussle

Extensibility is bad
- 2^n dialects
- (mitigation: tag 999)

Extensibility is good
- if properly framed into an extension point
- if implementation threshold remains low ("pluggable" approach)

Objectives for this meeting: CDE

- CDE
  - Complete NaN payload narrowing/widening discussion
    - Stick with RFC 8949 decisions; merge (PR #37)
  - Establish focus of CDE BCP
    - Do we need more text that is unrelated to CDE?
  - Clear way to progress — another WGLC?
    Compare -08 to -12

CDE: PR #37 (add to 3.1 CDE Constraints from Preferred Serialization)

Section 9.7 of [IEEE754] specifies an implementation-defined programming interface for accessing non-zero NaN payloads, the getpayload/setpayload functions. (A version of these is also slated for inclusion in the next revision of the C language.)
When using these functions, it is important that their effects are specific to the representation size of the floating point values they are applied to (e.g., half, single, or double precision).
The representation size for interchange will be chosen by Preferred Serialization for each value, which may not always be the size that was intended for the use of getpayload/setpayload.
A good way to handle this diversity is, upon decoding, to widen the representation size of all NaNs to a common size, often double precision ([IEEE754] binary64), before applying getpayload/setpayload.

CDE: Focus

- Objective: Single document that points out best practice for deterministic encoding of CBOR
  - CBOR application designer (e.g., in IETF or other SDO)
  - implementer that still needs to add CDE (e.g., Python cbor2)
- Non-objectives:
  - General treatise about creating "profiles" (subsets) of CBOR
  - General treatise about serialization

Objectives for this meeting: Packed CBOR

- Packed
  - Decide A=12/B=8/C=8
  - See Martine's DNS results, hackathon results
  - Prepare some final interop work
  - ➔ WGLC ~ end of August

Packed CBOR: Hackathon results

config (Cortex M4)	LOC (cloc)	code space (B)	context (B)
CBOR base	511	1826	24
" + packed-shared	804	3482	48
" + " + integration	900	3926	56

- WIP NanoCBOR implementation of shared item Packed CBOR
  - generic decoder aimed at highly constrained devices
  - "pull" decoder; decoder context per active CBOR container
- during hackathon: added support for splicing integration tag
  - record recursion level of integration tag in decoder context
  - only actually advance application-level decoder when all integration tag content has been processed

loose definition (= high flexibility) of integration tags:
requires special handling for specific semantics
➔ need to extend decoder context

Packed CBOR: A=12/B=8/C=8

Various tests with small data (DNS, …)

A: 12 seems to work in our examples and is conservative
B/C: 8 seems to work well, too

Do we still need to be stuck on these numbers
or can we continue?

Packed CBOR: Preparing the WGLC revision

More interop with integration tags

— Should that testing already include draft-amsuess-cbor-packed-by-reference-04 testing?

Address remaining issues (3 editorial, 1 technical):

— #20 Avoid invalid unpack results with 1112

Aim for WGLC ~ end of August