CBOR (RFC 7049) bis

Concise Binary Object Representation

Carsten Bormann, 2019-03-27

Take CBOR to STD

- **Do not**: futz around
- **Do**:
  - Focus on interoperability
  - Make needed improvements in specification quality

2019-11-04, <draft-ietf-cbor-7049bis-09.txt>

- Some 29 issues closed since IETF105
- WGLC started 2019-11-14, ending on **Thursday, 2019-12-12**.

Levels of Errors #45

- (not) well-formed — CBOR Syntax
  - Error: Not recoverable (outside diagnostic tools)
  - See also Appendix C (pseudocode)
- (not) valid — CBOR Semantics
  - Error: Presentable to the application in principle
- (not) expected — Application Syntax and Semantics
  - This is often expressed in CDDL

Generic Decoder

Other significant

- Appendix G: Well-formedness errors and examples
- #104 avoid fuzzy concept of “strict mode”;
  #122 avoid painting a “CBOR firewall” concept
- Tighten/clarify JSON-to-CBOR conversion issues
- Bug fix in in well-formed pseudocode (indefinite)
- Validity — next slide

Validity

- Distinguish basic validity (UTF-8, map keys) from tag validity
- Don’t assume that all generic decoders will do all possible validity checking — impossible for new tags, anyway

Remaining issue: #63

- What should be the onus on application protocol definitions and generic decoding libraries with respect to **duplicate map keys**?
- Proposal: No change.
  - Do not require all decoders to be validating, so can’t have a “MUST error out”.
  - Many decoders just silently discard duplicates (in varying ways), so application has little control
- Application can still require validity checking from their generic decoders, if really needed

CDDL

Nach dem Spiel ist vor dem Spiel
(After the game is before the game)

Next steps on CDDL
(RFC 8610)

draft-bormann-cbor-cddl-freezer

- Collected items that were not done for CDDL 1.0
- Can be thawed now
- What should we pick up?
- Let’s prioritize today

(0) Easily done using CDDL 1.0 extension points (control ops)

- computed literals (base = 400   a = base + 4)
- embedded ABNF

(0.1) computed literals

- `Zwei = 1 .plus 1`
- `Dogfood = “dog” .cat “food”`
- Proposal: `.plus .minus .cat` for now

(0.2) ABNF

- .abnf: control operator on text strings
- `Number = text .abnf “1*(%x30-39)”`

-
  ```
  Number = text .abnf (“number” .cat myabnf)
  myabnf = ‘
    number = 1*DIGIT
    DIGIT = %x30-39
  ‘
  ```
  (little trick: use byte string notation, as that allows newlines)
- Careful: need ABNF both for bytes and for characters (codepoints); proposal: .abnfbyte and .abnf

(1) Extend the **function** of CDDL

- Today: CDDL specification is a predicate on a CDDL instances, matches? ➔ true/false
- Could return more information, cf. PSVI (post schema-validation instance) in XML
  - E.g., defaulting
  - E.g., semantic augmentations
  - E.g., transformations

(2) Extend the **Expressiveness** of CDDL

- Cuts (e.g., for whole map members)
- Co-occurrence constraints (next slide)

(2.1) Co-occurrence constraints

- Predicates
- Pointers/Selectors

```
session = {  …   timeout: uint,  … }

other-session = {

  timeout: uint .lt [somehow refer to session.timeout],

}
```

(3) Syntactic Sugar

- tag-oriented literals — dt’2019-07-21T19:53Z'
    - ➔ transformations at the specification level
- regular expression literals

(4) CDDL in the large

- Module superstructure
  - Namespacing
  - Import/Export (relating to URIs?)
  - Versioning
- Variants (think #ifdef)

(99) Using CDDL for JSON and CBOR

- Support embedded JSON: .json operator (no-brainer)
- Maintain a single specification for both JSON and CBOR serialization: requires **variants**
- Separate issue: Enable use of JSON for CDDL representation, enabling tool interoperation (“CDDLJ”, next slides)

Alternative Representations (1)

```
cddlj = ["cddl", +rule]
rule = ["=" / "/=" / "//=", namep, type]
namep = ["name", id] / ["gen", id, +id]
id = text .regexp "[A-Za-z@_$](([-.])*[A-Za-z0-9@_$])*"
op = ".." / "..." /
  text .regexp "\\.[A-Za-z@_$](([-.])*[A-Za-z0-9@_$])*"
namea = ["name", id] / ["gen", id, +type]
type = value / namea / ["op", op, type, type] /
  ["map", group] / ["ary", group] / ["tcho", 2*type] /
  ["unwrap", namea] / ["enum", group / namea] /
  ["prim", ?(0..7, ?uint)]
group = ["mem", null/type, type] /
  ["rep", uint, uint/false, group] /
  ["seq", 2*group] / ["gcho", 2*group]
value = ["number"/"text"/"bytes", text]
```

Alternative Representations (2)

```
labeled-values = {
  ? fritz: number,
  * label => value
}
label = text
value = number
```

➔

```
["cddl",
 ["=",
  ["name", "labeled-values"],
  ["map",
   ["seq",
    ["rep", 0, 1, ["mem", ["text", "fritz"], ["name", "number"]]],
    ["rep", 0, false, ["mem", ["name", "label"], ["name", "value"]]]]]],
 ["=", ["name", "label"], ["name", "text"]],
 ["=", ["name", "value"], ["name", "number"]]]
```

Should there be a CDDL roadmap WG document?

- Could adopt something like -freezer as WG document
- No intent to ever publish as an RFC
- But an “official” document with (at least a snapshot of) directions that are moving towards consensus
- Document the priorities