There is no escaping
Background: CBOR-associated languages

— CBOR = representation and interchange format (binary, concise, efficient)
  — low-level visualization in text as cbor-pretty (hex with comments)

Two associated textual languages:

— EDN (cbor-diag) ➔ examples, diagnostics
  — Text form for single instance (item/sequence), convert back and forth (cbor.me)
  — Derived from JSON, made more useful for humans, added binary, tags, ... 
— CDDL ➔ specification, validation
  — Describe specific data model (grammar)
  — Inspired by ABNF, can describe JSON, CBOR, CSV*
Aside: EDN ("cbor-diag") vs. hexdump ("cbor-pretty")

"cbor-pretty" form

```
a4                                   # map(4)
01                                # unsigned(1) (=AS)
78                                # text(28)
636f6170733a2f2f61732e657861     # "coaps://as.example.com/token"
6d7066c652e636f6d2f746f6b656e
05                                # unsigned(5) (=audience)
76                                # text(22)
636f6170733a2f2f72732e657861     # "coaps://rs.example.com"
6d7066c652e636f6d
09                                # unsigned(9) (=scope)
7254656d7043
18                                # text(6)
27                                # "rTempC"
45                                # text(39) (=cnonce)
e0a156bb3f
```

(annotated hexdump)

"cbor-diag" (EDN) form

```
{
   / AS / 1 : "coaps://as.example.com/token",
   / audience / 5 : "coaps://rs.example.com",
   / scope / 9 : "rTempC",
   / cnonce / 39 : h'e0a156bb3f'
}
```

/(RFC 9200)/
## Agenda today

<table>
<thead>
<tr>
<th>CBOR</th>
<th>tags</th>
<th>EDN</th>
<th>CDDL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>edn-literal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e-ref</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cddl-more-control</td>
</tr>
<tr>
<td></td>
<td>cbor-packed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cbor-cde:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDE, &quot;Basic Serialization&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cbor-det:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>explain Deterministic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cbor-numbers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>explain numbers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cddl-modules</td>
</tr>
</tbody>
</table>
Diagnostic Notation (EDN)

— use EDN in tools (CI, diagnostics) and in documents, not to define a new interchange format
— Base: (Interoperable) JSON text; any JSON is EDN text.

— + (binary) byte strings:
  h'...hex...'  b64'...base64...
  'text'
— + tags: nnn(content)
  — e.g., 18(...COSE Sign1...)
— + general map keys (not just strings:
  — numbers, tags, arrays, ...)

```json
{
  60123 : { /* last-event (SID 60123) */
    47(60200) : { /* event-port-fault (SID 60200) */
      1 : "0/4/21", /* port-name (SID 60201) */
      2 : "Open pin 2", /* port-fault (SID 60202) */
    },
  }
} /* example from RFC 9254 */
```
EDN: Old and New Extensions for Usability

- embedded CBOR: `<<...>>` for a byte string of encoded CBOR

```
96([ / COSE_Encrypt /
   / protected / h'a10101',
]

96([ / COSE_Encrypt /
   / protected / << {
     / alg / 1: 1 / AES-GCM 128 /
   } >>
)] >>, /... RFC 9052 /
```

- readability: comments `/.../`, **new**: `#...`
- **new**: allow final comma in map/array: `[1, 2, 3, ]`
- **new**: comma is optional everywhere
- **new**: application-oriented literals:
  - `dt'2024-03-22T05:00:00Z' ➔ 1711083600`
  - `ip'192.0.2.42' ➔ h'c000022a'`
1. CBOR: Tags are extension point, “plugins”
   — EDN can't write tag semantically, just structurally
   — Idea: Have "plugins" for EDN literal formats as well
   — Based on byte string syntax: prefix'text'
2. Opportunity: General cleanup/usability enhancements
3. Opportunity: Answer frequent request: ABNF
EDN extension points: Adding External References to EDN

**e''**: accessing CDDL information

text in `e''` refers to CDDL names mapped to numbers via a CDDL module

draft-ietf-cbor-edn-e-ref-00
EDN extension points: Entering CRIs as URIs/IRIs

cri'': notating a CRI in URI form or IRI form

cri'https://example.com/bottarga/shaved'
→
[-4, ["example", "com"], ["bottarga", "shaved"]]

draft-ietf-core-href-16
Mapping application-extensions to ABNF

RFC 8949/8610 did not provide ABNF, just English

➔ new ABNF for edn-literals (derived from implementation)

```
app-string = app-prefix sqstr
app-prefix = lcalpha *lcalnum ; including h and b64
            / ucalpha *ucalnum ; tagged variant, if defined
sqstr = "'" *single-quoted "'
single-quoted = unescaped
              / DQUOTE
              / "\\" "'
              / "\\" escapable
```
Making application-extensions pluggable

Overall syntax for all application-oriented literals: prefix 'text', same as for h''/b64''

Deployability: Parser needs way to handle unknown prefixes (Section 3.1: can wrap literal text in tag 999)

per-prefix syntaxes: pluggable, in separate ABNF:

```
app-string-cri = IRI-reference
    ; copy and fix more stuff from RFC3986/3987
```
Provide separate ABNF per application-extension prefix

• per-prefix grammar applied in a secondary processing step
• enables deployability tag 999: 999(prefix, parsed text)

Alternative: Add each application-extension to base ABNF

— Each application-extension needs to define its own string parsing and escaping (and likely ABNF for that) consistency unlikely; mistakes likely
— base ABNF changes each time with adopting a new application-extension; cannot really do "plugins"
The discussion

WG specification uses pluggable architecture as described (extensions each with their own grammar based on the common de-escaped text — "two-layer" approach)

PR #49: Replace this with new "single-layer" approach
• each of the four app-prefixes in this document defines its own single-quoted string syntax
• each new one will need to do this in a compatible way

(lots more detailed discussion on the mailing list)
Pros and cons

one-layer:

— one-layer approach more familiar to early ABNF users
— will seem more familiar to implementers of other text representations
— easier sharing (but also more complexity) of ABNF rules

two-layer:

— true pluggables (runtime addition in real platforms)
— isolation between pluggables, between pluggable and base
— can use existing ABNF directly (e.g., cri), as escaping handled by base layer
— multiple layers bring separation of concern (IP/TCP, SIP/SDP, ...)

Concise Data Definition Language (CDDL)

draft-ietf-cbor-update-8610-grammar:
• housekeeping, ABNF fixes; non-literal tag numbers
• in RFC editor queue (EDIT)

draft-ietf-cbor-cddl-more-control
• Additional control operators, exercising explicit CDDL extension point
• another iteration like RFC 9165

draft-ietf-cbor-cddl-modules
• Support composition of CDDL model from multiple files
example use of .printf: \texttt{"0x1267: 0x1.46p+6"}
my-label = text .printf (\texttt{"0x%x: %a"}, 4711, 81.5)

Example from \texttt{draft-fft-rats-eat-measured-component-02}:

; direct:
$\texttt{measurements-body-cbor} /= \texttt{mc-cbor}
mc-cbor = bstr .cbor measured-component

; tunnel via JSON!
$\texttt{measurements-body-cbor} /= \texttt{tstr .b64u mc-json}
mc-json = tstr .json measured-component

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>.b64u, .b64c</td>
<td>Base64 representation of byte strings</td>
</tr>
<tr>
<td>.b64u-sloppy, .b64c-sloppy</td>
<td>(sloppy-tolerant variants of the above)</td>
</tr>
<tr>
<td>.hex, .hex1c, .hexuc</td>
<td>Base16 representation of byte strings</td>
</tr>
<tr>
<td>.b32, .h32</td>
<td>Base32 representation of byte strings</td>
</tr>
<tr>
<td>.b45</td>
<td>Base45 representation of byte strings</td>
</tr>
<tr>
<td>.decimal</td>
<td>Text representation of integer numbers</td>
</tr>
<tr>
<td>.printf</td>
<td>Printf-formatted text representation of data items</td>
</tr>
<tr>
<td>.json</td>
<td>Text representation of JSON values</td>
</tr>
<tr>
<td>.join</td>
<td>Building text from array of components</td>
</tr>
</tbody>
</table>

Table 1: New control operators in this document
2024-03-02 WGLC draft-ietf-cbor-cddl-more-control-03
2024-03-15 completed, ➔
–04: Leave deterministic encoding to draft-ietf-cbor-det
–05: clarify strictness, fix .join text
–06: nicer Table 1
Shepherd report appears to be ready

it?
— Data items can have considerable redundancy
— Method 1: deflate, brotli, zstd:
  Apply (byte-wise) data compression on encoded representation
  — Good compression
  — Relationship to uncompressed completely lost
    ➔ decompression = copy step
— Method 2: Apply compression at the data model level ("packing")
  — Consumer can directly work out of stored packed representation
  — ➔ Much more accessible to constrained implementation
Current draft essentially stable:

1. Stable: Reference set (tags, simple values) in place as **stand-ins** for referenced data

2. Reference semantics: (a) to **shared** (✔) or (b) to **arguments** (concatenation ✔ + many new ideas)
   ➔ **Function tags** as extension point; ✨ batteries included

3. **Table-building tags** build tables for references to refer to
   ➔ could be application-specific, extension point; ✨
draft-ietf-cbor-packed: Status

One issue open: representation of unpacking errors #20
Various PoC implementations

— drafts now starting to rely on cbor-packed:
  5 normative references so far
— Extensions/additional batteries, e.g.:
  — draft-amsuess-cbor-packed-by-reference
CBOR-Packed Documents

draft-ietf-cbor-packed-12: Packed CBOR
2024-03-02 I-D Exists: Waiting for WG Chair Go-Ahead

Related:

draft-amsuess-cbor-packed-by-reference-02: Packed CBOR: Table set up by reference -- 2024-03-04 I-D Exists
(related paper acceptance 2024-09-12)
Deterministic Encoding:

— **Defined** by RFC 8949, Section 4.2
  Based on Preferred Serialization (4.1)
— Left considerable *leeway* for applications (4.2.2)

*generic* deterministic encoder not fully standardized
CDE: approach

Nail down common deterministic encoding (CDE) draft-ietf-cbor-cde-05 (now fully stable)

Push leeway up one layer (application/Application Profile)

Discussion 1:
• Is there a need for Application Profiles?
  ➔ Yes, that possibility is part of the message.
    (CDE as narrow waist instead of multiple app-DEs)
Misusing CDE just for variability reduction

Dealing with partial (constrained) decoder implementations:
• reduce encoder choice (variability)

CDE does this perfectly
CDE has one expensive component: map ordering

Just preferred serialization is too weak (allows indefinite)

CBOR ➔ Preferred Serialization ➔ Basic Serialization ➔ CDE ➔ (can add app profiles on top, no change to CDE)
Editorial questions

Discussion 2:
• Do we need a separate document just for Basic Serialization?
  ➔ No, the technical content is two lines (preferred + definite)
  ➔ exactly one normative document about encoding

Next: decide discussions, one more editorial round, WGLC
— draft-bormann-cbor-det-03
explainer for deterministic encoding in general (including application layer)
• keeps CDE specification itself simple

— draft-bormann-cbor-numbers-00
explainer for the arcana of the number systems supported by CBOR
• read before det explainer ➔ also keeps CDE specification simple
• generally useful for implementers (who often are weak on numbers)

Can adopt at leisure, or decide to push to ISE
Within a CDDL project:

— Construct a project CDDL from multiple files
  (;# import * from)

— Reference existing CDDL as libraries
  (;# import [foo from])

— Optionally put imported CDDL into a namespace
  (...as)

"modules" are the core addition in "CDDL 2"
import automatically what is needed

import rfc9052

CWT-cnf = {
    (1: COSE_Key) //
    (2: COSE_Encrypt / COSE_Encrypt0)
}

CWT-cnf = {1: COSE_Key // 2: COSE_Encrypt / COSE_Encrypt0}

COSE_Key = {
    1 => tstr / int,
    2 => bstr,
    3 => tstr / int,
    4 => [+ tstr / int],
    5 => bstr,
    * label => values,
}

COSE_Encrypt = [
    Headers,
    ciphertext: bstr / nil,
    recipients: [+ COSE_recipient],
]

COSE_Encrypt0 = [
    Headers,
    ciphertext: bstr / nil,
]

label = int / tstr
decrypt = any

Headers = {
    protected: empty_or_serialized_map,
    unprotected: header_map,
}

COSE_recipient = [
    Headers,
    ciphertext: bstr / nil,
    ? recipients: [+ COSE_recipient],
]

empty_or_serialized_map = bstr .cbor header_map / bstr .size 0

header_map = {
    Generic_Headers,
    * label => values,
}

Generic_Headers = {
    ? 1 => int / tstr,
    ? 2 => [+ label],
    ? 3 => tstr / int,
    ? 4 => bstr,
    ? (5 => bstr // 6 => bstr),
}
Namespacing: as . . .

```python
;m import rfc9052 as COSE

CWT-cnf = {
    (1: COSE.COSE_Key) //
    (2: COSE.COSE_Encrypt / COSE.COSE_Encrypt0)
}
```

Some RFC 9052 naming was already trying separation (COSE_)
But other rules in RFC 9052 are not:
(label, values, empty_or_serialized_map)
→ namespacing these can help avoid name conflicts
Modules: Separation of concerns

— Make use of seamless CDDL1 compatibility
— Continue using variety of CDDL1 implementations: zcbor (cddl-codegen), anweiss cddl, early cddl tool, ...
— Make namespace/import processing highly inspectable
— Note that implementations are free to integrate this
— or simplify their intake by making use of other CDDL processor services (e.g., degenericizing) as well
cddlc as a preprocessor

cddlc -2tcddl input.cddl > completed.cddl

Can often be used in a pipeline:

cddlc -2tcddl input.cddl | cddl - gp 10
cddlc -2tcddl input.cddl | cddl - vp instance.cbor
---

- `# import *` does a wholesale inclusion (all rules) can trigger "unused rule" warnings
- `# import` operates on "undefined list"
  - only rules that satisfy undefined list are imported
  - these rules contribute to list: transitive closure

... as both can operate on names of imported rules
module: built from referenced CDDL files

— referenced file is first import-processed on its own
— then importing from resulting rules
— processing "in isolation": the referencing context does not change the meaning of a referenced module

➔ indirectly imported rules are logically present in the imported module
➔ can then be referenced, as well
module targets from where?

[✓] CDDL_INCLUDE_PATH, default ».:«
  • . is current directory
  • empty string at the end: curated "batteries included"
    (RFCs included in gem; possibly other canonical CDDL)

☐ TODO: scraping?
  • IANA? (unstable XML)
  • I-D draft-* (always via XML)
  • others? github? manufacturer sites?
remaining open issue: sockets

full.cddl:

;# import a
;# import b

x = [foo-a, foo-b]

a.cddl:

foo-a = tstr
$data /= int

b.cddl:

;# import c

foo-b = [+ $data]
$data /= float
$data /= true

c.cddl:

;# import RFC9052

$data /= bytes .b64u COSE_Sign1
sockets: rough solution

— In import, keep candidates ($data and dependencies)
— When a socket is imported in another module: add all candidates already seen for the socket
— *Somehow* handle namespaces graciously as well

SMOS + SMOP
— YANG is XML, so fundamentally text-based
— does have binary type, encoded as base64 classic in XML/JSON and byte string in YANG-CBOR
— draft-ietf-netconf-crypto-types-28 uses binary throughout

Issue: bulky textual data in RFC 6991/bis remain:
— date/time (RFC 3339 style)
— IP address/prefix (nnn.nnn.nnn.nnn and RFC 4291/5952)
Efficient YANG-CBOR: Stand-

CBOR Tags: 1 (date/time), 52/54 (IP address/prefix)
"Stand-In": Can replace text with tags, where appropriate

— Define equivalence on the textual level (stand-in for text string)
— Schema-driven encoder in practice
— [ ] For canonically encoded items only?
— [ ] Zone-ID exceptions?
Efficient YANG-CBOR: Stand-Ins

— Main work needed:
  — how to announce capability?
    — Media type parameter?
    — Library?
  — A proxy can change the capability
    — how to agree to rely on concise representations?
      — Constrained implementations do not want to do the text-based YANG types ➔ can't (default), can, need × send/receive
  — which WG to do that work: cbor, core, netmod, netconf?