CBOR

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 1c                                # text(28)
      636f6170733a2f2f61732e657861
      6d706c652e636f6d2f746f6b656e      # "coaps://as.example.com/token"
   05                                   # unsigned(5) (=audience)
   76                                   # text(22)
      636f6170733a2f2f72732e657861
      6d706c652e636f6d                  # "coaps://rs.example.com"
   09                                   # unsigned(9) (=scope)
   66                                   # text(6)
      7254656d7043                      # "rTempC"
   18 27                                # unsigned(39) (=cnonce)
   45                                   # bytes(5)
      e0a156bb3f                        #
```

"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)/
```

(annotated hexdump)

Agenda today

CBOR	tags/apps	EDN	CDDL
	cbor-packed		
		edn-literal	
cbor-cde: Common Deterministic Encoding			
			cddl-modules
	dns-cbor		

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: (next slide)

— + tags: `nnn(content)`
  — e.g., `18(...COSE Sign1...)`

— + general map keys (not just strings: — numbers, tags, arrays, ...)

```
{
  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: Status

- Started out as draft for adding extension point to EDN: application-oriented literals
- Proposal to add ABNF:
  Done 2023-07-10: draft-ietf-cbor-edn-literals-01
- 2024-05-03: Publication requested
- Proposal to roll up text from RFC 8949 and 8610
  Done 2024-11-03: draft-ietf-cbor-edn-literals-13

Notating Binary Strings

Just enter the text:
`"..."` -- literal for a text string (UTF-8, major type 3)
`'...'` -- literal for a byte string (major type 2)
Escapes (`\"`, `\u2318`) for things not easy to put in as text

But often want to process EDN text to get byte string
➔ prefixed single-quoted string:
`h'...hex...'` `b64'...base64...'`

Unusual: Comment syntax in prefixed literals

prefixed strings

Use prefixed single-quoted string syntax for:

- byte strings in base16/base64 encoding:
  `h'...hex...'` `b64'...base64...'`

- new: application-oriented literals:
  `dt'2024-03-22T05:00:00Z'` ➔ `1711083600`
  `ip'192.0.2.42'` ➔ `h'c000022a'`

application-oriented literals

Register a prefix (extension point)
Define: prefix name, semantics:
• what does prefix do with text string argument

Syntax: **prefix'argument'**
(looks syntactically like `h'3c9f'`)

Idea: Handle all prefixed strings in the same way

EDN extension points: Adding External References to EDN

e'': accessing CDDL information

```
96([ / COSE_Encrypt /                    96([ / COSE_Encrypt /
     / protected / << {                       / protected / << {
       / alg / 1: 1 / AES-GCM 128 /     →       e'alg': e'AES_GCM_128'
     } >>,          /... RFC 9052 /           } >>,          /... RFC 9052 /
```

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

prefixed strings 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
                / ucalpha *ucalnum
sqstr           = "'" *single-quoted "'"
single-quoted   = unescaped
                / DQUOTE
                / "\" "'"
                / "\" escapable
```

Making application-extensions pluggable

Same overall syntax for all prefixed strings
prefix'text', where **'text'** is single-quoted string

Extension point (all extensions treated the same):
take **content** (de-escaped) of single-quoted string
as input to prefix-specific processing

Making application-extensions deployable

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

per-prefix syntaxes:
pluggable, each with their own grammar

```
app-string-cri = IRI-reference
```

Contention Point

Instead of pluggables, couldn't each extension define its own ABNF that is then edited into the 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 document uses pluggable architecture as described (extensions each with their own grammar based on the common de-escaped text content — "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)

A practical approach

— Keep overall ABNF grammar separated from grammars for prefixed string content

— Provide a mechanical translation from per-prefix grammars to a squashed ABNF

```
HEXDIG = DIGIT /
         "A" / "B" / "C" / "D" / "E" / "F"
DIGIT  = %x30-39
```

```
qHEXDIG = qDIGIT /
          "A" / "B" / "C" / "D" / "E" / "F" /
          (ULZ ("4"/"6") %x31-36 "}")
qDIGIT  = %x30-39 /
          (ULZ ("3") %x30-39 "}")
ULZ     = %s"\u{" *"0"
```

Potential tweaks (1)

(1) Provide separate content de-escaping rules for unprefixed and prefixed strings
(E.g., disallow \u0000 to \u007f etc. for prefixed)

— + simplifies single-level ABNF implementations for certain specific cases

— – requires user to remember different rules for the two cases

Could disallow \u0000 to \u007f everywhere?
– No longer JSON

Potential tweaks (2)

(2) Provide special content de-escaping just for traditional byte strings `h''` and `b64''`
(E.g., disallow most escapes just in these)

— + simplifies single-level ABNF implementations for these specific cases

— – requires user to remember different rules for the two cases

EDN: Proposal

1. Stick with untweaked, common syntax for string literals
2. Ship existing ABNF with EDN specification
3. Provide a tool to translate base ABNF combined with ABNF snippets for a number of prefixed strings into a single ABNF file

CDE: Common Deterministic Encoding

- Deterministic Encoding: The same data → the same encoded form
  - Need to choose "Preferred Encoding"
  - Separating out map ordering (expensive)
- Defined by Section 4.2 of RFC 8949
  - Leaves several choices open to application

CDE: Common Deterministic Encoding

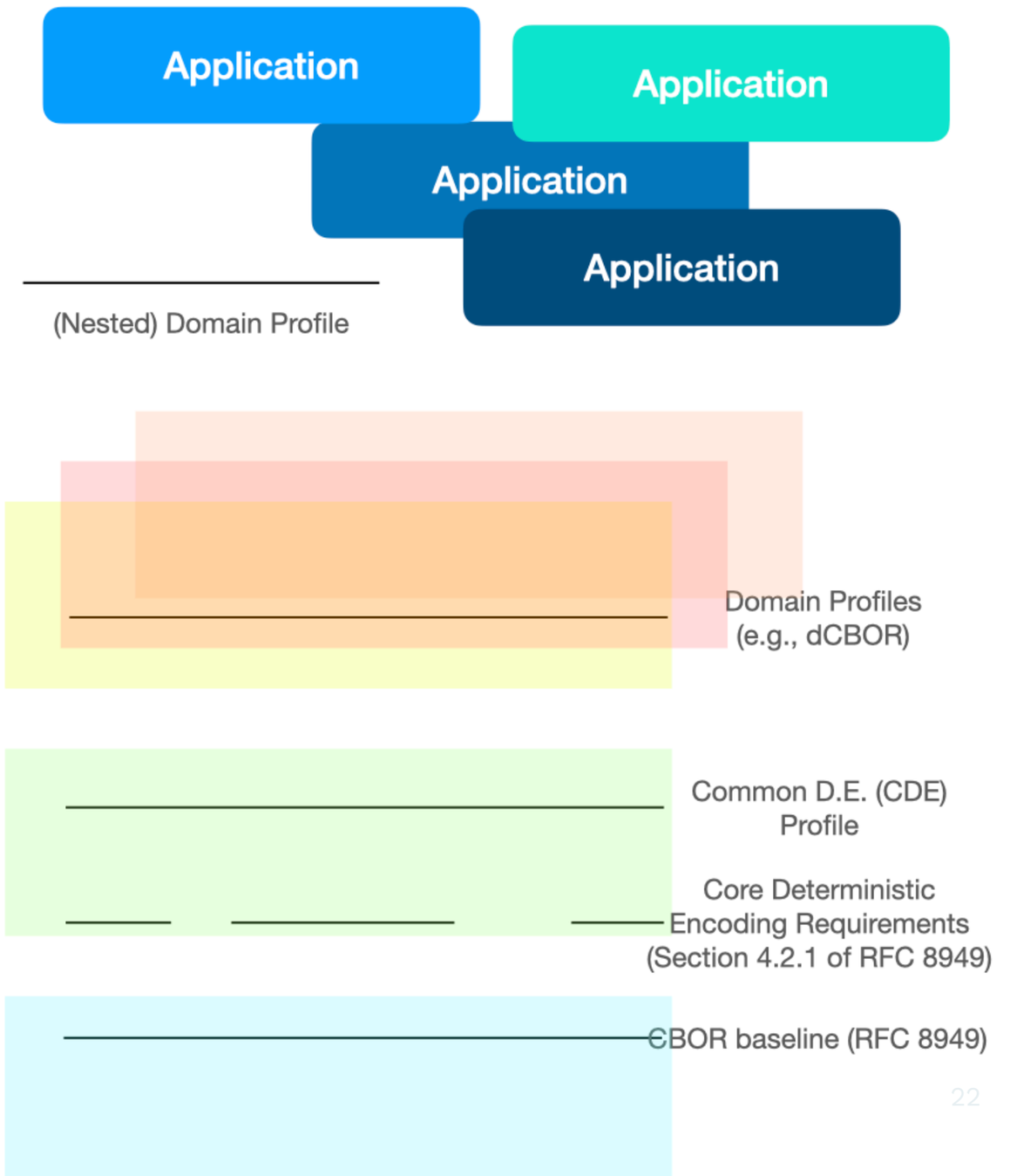
Recent advance:
Split "deterministic encoding" into:

— Common CBOR Deterministic Encoding Profile (CDE)

concerned with mapping: bytes ⟷ data items at data model level

— Application-level Deterministic Representation (ALDR) rules

concerned with mapping: data items at different data model levels (generic CBOR ⟷ domain specific)



CDE: Status

- 2023-11-27 WG document
- lots of editorial tweaking, merged PRs
- 2024-10-16 draft-ietf-cbor-cde-06
  - introduces ALDR discussed at 2024-10-16 interim
  - references cbor-numbers for NaN appendix
- stale PR #10?

Would like to WGLC this now.

draft-ietf-cbor-cddl-modules: Objectives

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"

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

— During import, keep each socket as a candidate (`$data` and dependencies)

— When a socket is imported in another module: add all candidates already seen for the socket

— Define how to handle namespaces for sockets (`import ... as`)

SMOS + SMOP