Tutorial block: "Using CBOR in specifications"

15:00–15:30

- EDN: (Extended) Diagnostic Notation: readable examples, diagnostics
  - new individual: e-ref, draft-numbers -- adopt?
- CDDL: Concise Data Definition Language, "grammar"
  - recent WG work: more-control and cddl-modules

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*

Diagnostic Notation (EDN): draft-ietf-cbor-edn-literals

Base: (Interoperable) JSON text; any JSON is EDN text as well.

Basic additions for additional CBOR data items:
- binary strings encoded: `h'...hex...'` `b64'...base64...'` `'text'`
- 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 /
```

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)

EDN: Old and New Extensions for Usability

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

draft-ietf-cbor-edn-literals-08

(historic draft name):
has all the above, plus finally ABNF grammar for EDN

— ABNF for use in tools (CI including for specs, diagnostics), not to define a new interchange format

Passed WGLC; will be with IESG soon
Has extension points

EDN extension points: Adding External References to EDN

e'': accessing CDDL information

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

```
96([ / COSE_Encrypt /
    / protected / << {
      e'alg': e'AES_GCM_128'
    } >>,       /... RFC 9052 /
```

text in `e''` refers to CDDL names

e": gm-admin example

bad (wrong!)

```
{
    "group_mode" : true,
    "gp_enc_alg" : 10,
          "hkdf" : 5
}
```

first add CDDL:

```
group_mode = 33
gp_enc_alg = 34
hkdf = 31
HMAC-256-256 = 5
AES-CCM-16-64-128 = 10
```

Correct gmadmin.diag (with e''):

```
{
    e'group_mode' : true,
    e'gp_enc_alg' : e'HMAC-256-256',
          e'hkdf' : e'AES-CCM-16-64-128'
}
```

In cddlc, CDDL for e'...' is found via CBOR_DIAG_CDDL

```
$ export CBOR_DIAG_CDDL=gmadmin.cddl
$ diag2diag.rb -ae gmadmin.diag
{33: true, 34: 10, 31: 5}
```

Handling tentative label assignments in drafts

draft-bormann-cbor-draft-numbers-03

Label all tentatively assigned numbers with "CPA" in draft (CDDL, EDN!)
Clarify processing required at approval time

CDDL

```
problem-details = {
  ? &(title-CPA: -1) => oltext
  ? &(detail-CPA: -2) => oltext
  ? &(instance-CPA: -3) => ~uri
  ? &(response-code-CPA: -4) => uint .size 1
  ? &(base-uri-CPA: -5) => ~uri
  ? &(base-lang-CPA: -6) => tag38-ltag
  ? &(base-rtl-CPA: -7) => tag38-direction
  ; ...
  * (uint .feature "extension") => any
}
```

EDN

```
{
  / title-CPA /         -1: "title of the error",
  / detail-CPA /        -2: "detailed information",
  / instance-CPA /      -3: "coaps://pd.example/FA31",
  / response-code-CPA / -4: 128, / 4.00 /
  4711: {
     / ... /
  }
}
```

e'' reduces, but does not remove the need for CPA convention

```
{
  e'title'         : "title of the error",
  e'detail'        : "detailed information",
  e'instance'      : "coaps://pd.example/FA31",
  e'response-code': e'code400',
  4711: {
     / ... /
  }
}
```

```
CDDL for that:
    title = -1          ; CPA
    detail = -2         ; CPA
    instance = -3       ; CPA
    response-code = -4 ; CPA
    code400 = 128       ; 4.00
```

EDN extension points: Adding External References to EDN

ref": accessing EDN information

- Leave out large certificates etc. from spec examples
- make the EDN fit for CI by referencing from external

```
/ oemboot /         262: true,
/ dbgstat /         263: 2, / disabled-since-boot /
/ measurements /    274: [
                           [
                             121, / CoAP Content ID. A     /
                                  / made up one until one  /
                                  / is assigned for CoSWID /

                            / This is a byte-string wrapped /
                            / evidence CoSWID. It has       /
                            / hashes of the main files of   /
                            / the IoT OS.  /
                            h'a60063463613234350c
                              17016d41636d6520522d496f542d4f
                              530d65332e312e3402a2181f724163
                              6d65204261736520417474657374657
                              7218210103a11183a318187161636d
                              655f725f696f745f6f732e65786514
                              1a0044b349078201582005f6b327c1
                              73b4192bd2c3ec248a292215eab456
                              611bf7a783e25c1782479905a31818
                              6d7265736f75726365732e72736314
                              1a000c38b10782015820c142b9aba4
                              280c4bb8c75f716a43c99526694caa
                              be529571f5569bb7dc542f98a31818
                              6a636f6d6d6f6e2e6c6962141a0023
                              3d3b0782015820a6a9dcdfb3884da5
                              f884e4e1e8e8629958c2dbc7027414
                              43a913e34de9333be6'
                           ]
                         ]
```

```
e'oemboot': true,
e'dbgstat': e'disabled-since-boot',
e'measurements': [
  [ e'meas-content-format',
    ref'measurement1.diag', ]
]
/ via draft-ietf-rats-eat-25 /
```

Concise Data Definition Language (CDDL)

draft-ietf-cbor-update-8610-grammar:
- housekeeping, ABNF fixes; non-literal tag numbers
- finished WGLC, will be with IESG soon

draft-ietf-cbor-cddl-more-control
- Additional control operators
- another iteration like RFC 9165

draft-ietf-cbor-cddl-modules
- Support composition of CDDL grammar from multiple files

draft-ietf-cbor-cddl-more-control

```
; example use of .printf: → "0x1267: 0x1.46p+6"
my-label = text .printf (["0x%x: %a", 4711, 81.5])
```

Example from draft-fft-rats-eat-measured-component-02:

```
; direct:
$measurements-body-cbor /= mc-cbor
mc-cbor = bstr .cbor measured-component

; tunnel via JSON
$measurements-body-cbor /= tstr .b64u mc-json
mc-json = tstr .json measured-component
```

Name	Purpose
`.b64u`, `.b64c`	Base64 representation of byte strings
`.b64u-sloppy`, `.b64c-sloppy`	(sloppy-tolerant variants of the above)
`.hex`, `.hexlc`, `.hexuc`	Base16 representation of byte strings
`.b32`, `.h32`	Base32 representation of byte strings
`.b45`	Base45 representation of byte strings
`.decimal`	Text representation of integer numbers
`.printf`	Printf-formatted text representation of data items
`.json`	Text representation of JSON values
`.join`	Building text from array of components
`.cbordet`, `.cborseqdet`	deterministically encoded CBOR data items, CBOR sequences

Table 1: New control operators in this document

draft-ietf-cbor-cddl-modules: Objectives

Within a CDDL project:

— Construct a project CDDL from multiple files (`;# include`)

— Reference existing CDDL as libraries (`;# import`)

— Optionally put included/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
values = 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 . . .

```
;# import rfc9052 as COSE

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

Note that RFC 9052 naming was already namespacing feebly (`COSE_`)
But other rules in RFC 9052 are not:
(`label`, `values`, `empty_or_serialized_map`)
So namespacing these can help avoid name conflicts

Namespacing: create local aliases

```
;# import COSE_Key, COSE_Encrypt, COSE_Encrypt0 from rfc9052 as COSE

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

— now namespaced as COSE.label etc.

— Alias rules added for given names

➔

```
CWT-cnf = {1: COSE_Key // 2: COSE_Encrypt / COSE_Encrypt0}
COSE_Key = COSE.COSE_Key
COSE_Encrypt = COSE.COSE_Encrypt
COSE_Encrypt0 = COSE.COSE_Encrypt0
COSE.COSE_Key = {
  1 => tstr / int,
  ? 2 => bstr,
  ? 3 => tstr / int,
  ? 4 => [+ tstr / int],
  ? 5 => bstr,
  * COSE.label => COSE.values,
}

...

COSE.label = int / tstr
COSE.values = any
COSE.Headers = (
  protected: COSE.empty_or_serialized_map,
  unprotected: COSE.header_map,
  )

...
```

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/i* 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
```

;# include vs. ;# import

— `;# include` does a wholesale inclusion (all rules)
can trigger "unused rule" warnings

— `;# import` operates on "undefined list"
- only rules that satisfy undefined list are included
- these rules contribute to list: transitive closure

… as : both can operate on names of included rules

module: referenced CDDL files

- referenced file is import/include processed by itself
  - before importing/including its rules
  - processing "in isolation": the referencing context does not change the meaning of a referenced module

➔ indirectly i*ed rules are logically present in the module
➔ can 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, see below)
- others? github? manufacturer sites?

YANG-CBOR Efficiency: yang-standin 15:30-15:55

— 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?

draft-ietf-cbor-packed

- 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

draft-ietf-cbor-packed

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 the 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:
  4 normative references so far
  (2 oscore, 1 coral, 1 dns-cbor)
— Extensions/additional batteries, e.g.:
  — draft-amsuess-cbor-packed-by-reference