CBOR@IETF116

- ℹ Usable Formal Methods pRG met yesterday
- ⚠ "Profiles"
-  Media Type for cbor-diagnostic
- 🕒 CBOR tags: draft-ietf-cbor-time-tag (tag 1001..)
-  DNS and CBOR (➔ Martine)
-  CDDL evolution (CDDL 2.0 and beyond)

⚠ CBOR/COSE/CDDL "profiles" ⚠

(mcnally-deterministic-cbor) application profile for deterministic representation of application information in CBOR data model and encoding

(cose-profiles) — agreement on which subset of COSE features they will use; new COSE header parameter for in-band signalling of profile information

...

➔
topic interim on "profiles"/"feature sets"/"common usage"?

application/json

application/awesome.v0+ld+json;

version=1.4.2.alpha.13;

profile=
"https://cbor.cool
/urn:uuid:5c46e432-...-7eaefce9378a
?version=web5
#version=coolest"

courtesy of Orie Steele

Media Type for EDN (cbor diagnostic notation)

CBOR diagnostic format is not an "interchange format"
It is still helpful to identify it in tool pipelines

Proposal: register `application/cbor-diagnostic`
Now in "Application Specific EDN literals" draft

Keep in this draft or register separately?
Can we ship EDN-literals (independent of core-href)?

🕐 draft-ietf-cbor-time-tag

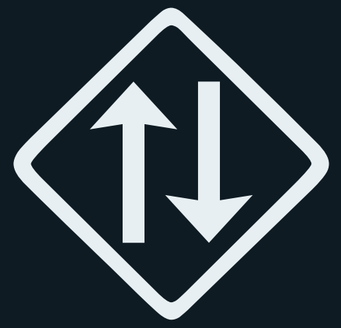
draft-ietf-cbor-time-tag

draft-ietf-cbor-time-tag-00 adopted 2021-05
"**no rush**": tags registered, in use in implementations
WGLC on –04 completed 2023-01-27; –05 submitted 2023-03-13
• adds requested CDDL definitions
• this should now address all WGLC comments.

Aiming for synchronized publication with
SEDATE WG Internet Extended Date/Time Format (IXDTF)
• SEDATE blocked on charter update; appears to be unblocking
➔ Still time to collect comments.

DNS and CBOR

lenders-dns-cbor



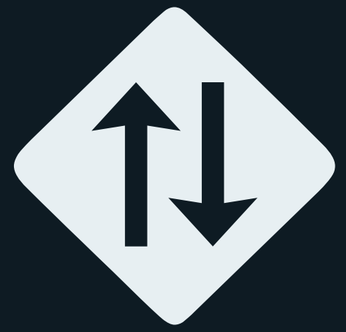
CDDL 2.0

CDDL 1.1 + 2.0 plan (standards track)

— Done before IETF 117: CDDL 1.1: Grammar fixes
draft-bormann-cbor-update-8610-grammar
Empty files (enabling CDDL 2), non-literal tags, errata fixes

— Done before IETF 117: Mid-2023 Parallel to CDDL 1.1: More control operators
draft-bormann-cbor-cddl-more-control
Additional control operators, another iteration like RFC 9165 (implemented)

— Done before IETF 118: CDDL 2.0:
draft-bormann-cbor-cddl-modules
Draft implementation already available for `import/include`



CDDL 2.5

CDDL "2.5" plan (standards track)

— To be done 2024:
CDDL 2.5: §3 of draft-bormann-cbor-cddl-2-draft
("annotations", functionality enabled by that)
➔ Lots of music, lots of fun.
Enables, e.g., §5 of draft-bormann-cbor-cddl-freezer
(co-occurrence)

Not on the main line of development:

— (I Mid-2023): §6 of draft-bormann-cbor-cddl-freezer
CDDL-in-JSON format(s) for interchange between tools

— (I, with 2.0) draft-bormann-cbor-rfc-cddl-models
(Builds standard collection of referenceable models)

— (S) §2.3/A.1 of draft-bormann-cbor-cddl-2-draft
literals; develop with draft-bormann-cbor-edn-literals

— (B) draft-bormann-cbor-draft-numbers
(BCP for handling assigned numbers during draft stage)

— (I/S?) draft-bormann-cbor-cddl-csv

↯ CDDL 1.1: Grammar update

Small language changes and fixes

— Non-Literal Tag Numbers

```
ct-tag<content> = #6.<ct-tag-number>(content)
ct-tag-number = 1668546817..1668612095
```

— Allow empty files

— Fix errata (byte string notation)

🛣 CDDL 1.1: More controls

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
`.json`	Text representation of JSON values
`.join`	Building text from array of components
`.cbordet`, `.cborseqdet`	deterministically encoded CBOR data items, CBOR sequences

⇅ CDDL 2.0: modules, composition

From CDDL 1.0: Concatenating files (which files? external)
to CDDL 2.0: explicit references!

➔ stay compatible with CDDL 1.0

```
;# import oid from RFC9090
; can now use
a = [oid]
```

CDDL 2.0: Module structure: (See 2023-02-08 interim)

Objectives:

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

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

— Optionally put i*ed CDDL into a namespace (…`as`)

— Optionally limit to specific names (…`from`)

CDDL 2.0: Status

Current feature set implemented in `cddlc` tool
Can be combined with other tools (e.g., `cddl`):

```
$ cddlc -2tcddl -icose=rfc9052 -scose.COSE_Key | cddl - gp
```

```
/$.start.$/ {1: "wobbegong", 2: /bstr/ h'65637461736961', 3: -181,
 4: ["candela"], 1035: /cose.values/ "unafraid",
 -205: /cose.values/ "steering", 2715: /cose.values/ "misobey",
 "Ahrimanian": /cose.values/ "skirp"}
```

Features not needed for standardization (🤔):

- Expanding generics
- "flattening" (introducing additional rule names for structure info)

⇅ CDDL 2.5: Update Processing model

RFC 8610: Validation (yes/no)
RFC 9165: add features (list of features used)

cddl tool gp/vp: "annotated" instance
not currently influenceable from model
➔ annotations similar to Relax_NG

Next step: transformation

CDDL 2.5: Using annotations

Annotations augment a rule name
Can introduce (invisible) rules to carry annotation

...syntax for adding attributes, e.g.,
`<< name: value >>`

Example: default values

```
? pizza-size: unit <<default: 35>>
```

Example:
co-occurrence constraints

```
session = {
  timeout: uint,
}

other-session = {
  timeout: uint  .lt [somehow refer to session.timeout],
}
```

Example: transformation specifications

```
length: uint <<scale: 10>>
```

Input: CBORPath?

YANG uses XPath for co-occurrence constraints
XPath 3.1: weird programming language, XML-faced

JSONPath: 2007 Gössner XPath replacement
JSONPath WG ➔ standardizing this now (WGLC soon)

CBORPath proposal
https://github.com/cbor/cbor.github.io/issues/90

⇅ CDDL 2.1: Cross-universe (IANA) references

```
cose-algorithm = int .iana ["cose", "algorithms", "value"]

//iana:registry[@id='algorithms']/iana:record/iana:value
```

➔ https://www.iana.org/assignments/cose/cose.xml

⇅ CDDL 2.x: ABNF is a lot like CDDL

ABNF = CDDL for flat sequences (of characters)
Integrated in CDDL via `.abnf`/`.abnfb`

CDDL 2.0:
Could provide many of the innovations for ABNF as well

Backup slides

Simple import (intrusively, from "library")

```
$ cddlc -2tcddl -
start = COSE_Key
;# import rfc9052
```

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

import as (namespaced from "library")

```
$ cddlc -2tcddl -
start = cose.COSE_Key
;# import rfc9052 as cose
```

```
start = cose.COSE_Key
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
```

Copy/Paste per explicit names in `include`

```
$ cddlc -2tcddl -
mydata = {* label => values}
;# include label, values from rfc9052
```

```
mydata = {* label => values}
label = int / tstr
values = any
```

Namespaced "as cose":

```
$ cddlc -2tcddl -
mydata = {* label => values}
;# include cose.label, cose.values from rfc9052 as cose
```

```
mydata = {* label => values}
cose.label = int / tstr
cose.values = any
```

Explicit name plus transitive closure: `import`

```
$ cddlc -2tcddl -
mydata = {Fritz: cose.empty_or_serialized_map}
;# import cose.empty_or_serialized_map from rfc9052 as cose

➔

mydata = {"Fritz" => cose.empty_or_serialized_map}
cose.empty_or_serialized_map = bstr .cbor cose.header_map / bstr .size 0
cose.header_map = {
  cose.Generic_Headers,
  * cose.label => cose.values,
}
cose.Generic_Headers = (
  ? 1 => int / tstr,
  ? 2 => [+ cose.label],
  ? 3 => tstr / int,
  ? 4 => bstr,
  ? (5 => bstr // 6 => bstr),
  )
cose.label = int / tstr
cose.values = any
```

Namespaced `import` with adding unnamespaced alias

```
$ cddlc -2tcddl -
mydata = {Fritz: cose.empty_or_serialized_map}
;# import empty_or_serialized_map from rfc9052 as cose
```

```
mydata = {"Fritz" => cose.empty_or_serialized_map}
empty_or_serialized_map = cose.empty_or_serialized_map
cose.empty_or_serialized_map = bstr .cbor cose.header_map / bstr .size 0
cose.header_map = {
  cose.Generic_Headers,
  * cose.label => cose.values,
}
cose.Generic_Headers = (
  ? 1 => int / tstr,
  ? 2 => [+ cose.label],
  ? 3 => tstr / int,
  ? 4 => bstr,
  ? (5 => bstr // 6 => bstr),
  )
cose.label = int / tstr
cose.values = any
```

Command line Control

```
$ cddlc -2tcddl -icose=rfc9052 -scose.COSE_Key
```

```
-icose=rfc9052 ➔

;# import rfc9052 as cose

-scose.COSE_Key ➔

$.start.$ = cose.COSE_Key
```

```
$.start.$ = cose.COSE_Key
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
```