CBAR/CBAPT: alternative to cbor-packed

What's new since January 22, 2025 (ietf-interim-2025-cbor-02)

CBAR/CBAPT: alternative to cbor-packed

CBAPT: CBOR & generic BLOB Atoms, Packing & Templating

Multi-purpose framework for CBOR (and not only) transformations.

- like Jinja2, but in and for CBOR

- like EXI for XML, but for CBOR (and may be dynamic)

In fact, very specialized language, like SQL (or may become so) or `jq` utility.

CBAR/CBAPT: alternative to cbor-packed

CBAR: CBOR and generic BLOBs by-Atom Reducing

Simplified subset of CBAPT for constrained implementations.

- e.g. for DNS: to replace cbor-packed, matching it's goals

Goals / Rationale

From most constrained to most powerful:

1. Semantic "packing" of CBOR message with ability to access it in-place without temporary memory buffers for decompression.
   - Constrained implementations can: consume and generate.
2. Reducing size of sent sensor data based on inter-packet templates - generalizing approach of SenML and YACTS/VTS.
   - Constrained implementations can: generate
3. Bring CBOR in size efficiency on-par to classic binary TLV protocols. An example is Netflow v9/IPFIX: packets are decoded according to template which is not fixed (constrained impls in p.2 above will usually hardcode template) but sent as separate packet relatively rare.
   - for full implementations: a small CBOR header with large `bstr` unpacked to normal CBOR by CBAPT decoder
4. Under exploration: an alternative to https://capnproto.org/rpc.html for multi-method calls on high-latency links (RPC with less round-trips).

One tag to rule them all

Tag #6.10 is THE tag of CBAR, used for everything, either alone or in combination with tags 63, 24 or Alternatives Tags (121-127, 1280-1400)

```
#6.10([commands])

commands = command / command +( ';' command )

command = cmd-name *argument
cmd-name = any  ; well, probably maps should be excluded
argument = any
```

What commands (last of them) return is substituted in place 10([...]). Other uses of tag #6.10 are actually shortcuts (but constrained implementations may hardcode).

E.g.

```
10(122(42))
```

is something like

```
 10(['with-dict', 1,
     10(['get-atom-value', 42])
])
```

```
10('some-bstr')
```

is something like

```
10(['template-process-cbar',
    'some-bstr''
])
```

```
10(122({"k1": "val1", "k2: 10(0)}))
```

is something like

```
10(['with-dict', 1, 'process-cbor',
       {"k1": "val1",
        "k2":  10(0) ; recursion
       }
])
```

CBAR: "template" is static, so just like cbor-packed

```
#6.10([simple-function, +atoms, CBAR-rump])
```

for example,

```
10([simple(0), "foo", "bar", "baz", [simple(0), 1], ...]
```

fills dictionary 0 with atoms to be then used on CBAR-rump, usage is simple: it will guess if it's CBOR or `bstr` or need deflate-unpacking first - "simple" is simple to use (`simple(1)` will fill dictionary 1, and so on to `simple(6)`).

Atoms accepted by simple function, e.g. form:

```
10([simple(0), "rgbValue", "rgbValueRed", "rgbValueGreen", CBAR-rump]
```

could be compressed by applying The Tag to atoms themselves:

```
 10([simple(0), "rgbValue",
    10(h'4B /#2.11/' + h'C0 /Atom 0/' + "Red"),
    10(h'4D /#2.13/' + h'C0 /Atom 0/' + "Green"),
    ...]
```

Note that `simple(N)` values here have their meaning on immediate level only - they are not interpreted inside compound arguments leaving them to applications. That is, `[simple(0), 1]` in first example above is not interpreted, it's up to application.

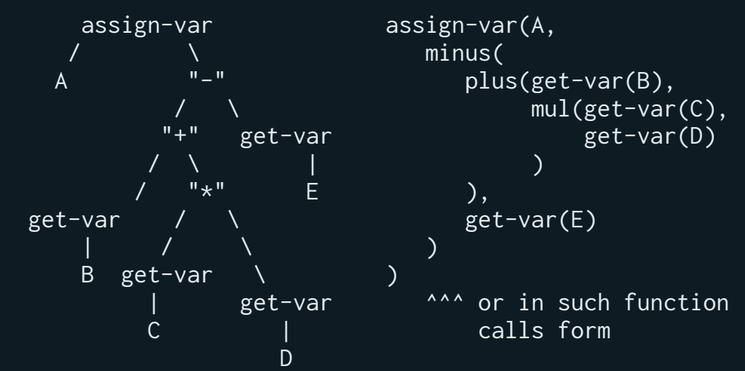
Thus, in contrast to cbor-packed, **CBAR/CBAPT don't require simple values registration "one and for all applications"!**

CBAPT background: Abstract Syntax Tree (AST)

In human-oriented language, expression like

```
A = B + C*D - E
```

could be represented in tree form like




```
assign-var(A,
   minus(
      plus(get-var(B),
           mul(get-var(C),
               get-var(D)
           )
      ),
      get-var(E)
   )
)
^^^ or in such function
    calls form
```

This maps naturally to CBOR data model

Of course reality needs some more syntax - problems:

1. Naive recursion will evaluate both "then" and "else" branches:

```
  if        so need a way for function to pass
 /  \       it's arguments to evaluator
then  else
```

2. We are processing CBOR: need to distinguish compound arguments (e.g. arrays) from code.

```
['assign-var', ['get-var-address', A],
  ['minus',
    ['plus',
      ['get-var-value', B],
      ['multiply',            ; what if not an arithmetic and
        ['get-var-value', C], ; you need an array as argument?
        ['get-var-value', D]
      ]
    ],
    ['get-var', E]
  ]
]
```

3. Compactness: one of goals is packing.

Stack machine?

At first stack machine was considered - as atoms MUST be 3+ character long, then 1-2 may be operations on stack: it's compact and it's simple to implement:

```
10(["foo", "bar", "baz", 2, '$', 'ld', rump])
```

But Forth approach had problems with variable-argument functions - in this example it is just one implicit function, but when you need several, you end up dancing with stack

```
..., 1, 'llength', 'ua', 0, 'ua', 1, 'ua', '(', 'ua', 'splice', ...
```

where 'ua' is `'unshift_arg'` to move top-of-stack (including '(' var-arg delimiter) right after '(':

And Yoda magister tells human for hard read/write it is

But we can use it's compactness due to suffix-notation also in prefix notation!

Solution - inverted Forth :)

— Tag 10 tells "execute me", otherwise it's just a CBOR argument.

— Simplest "parser" on top of "already lexed and parsed" tree:

```
10([simple(0), 'ld', '.', "example.org/dict", '+', 2, '$', 1, CBAR-rump])
```

is really

```
 10([simple(0),
     10(['load-cbor-document',
         10(['strconcat', "example.org/dict",
            10(["add", 2,
                10(["get-scalar-variable-value", 1])
            ])
         ])
     ]),
     CBAR-rump
 ])
```

given preambula somewhere in standard library with abbreviation and argument count like

```
'parse-alias', 'ld', 1, 'load-cbor-document'
'parse-alias', '.',  2, 'strconcat'
'parse-alias', '+',  2, "add"
'parse-alias', '$',  1, "get-scalar-variable-value"
'parse-alias', '@',  1, "get-unwrapped-list-from-array-variable"
```

will load atoms from CBOR Sequence at "example.org/dict5" if $1 variable contained 3 (and integer auto-converted to string on concatenation), or if document is CBOR array, insert '@' before 'ld'.

```
; if don't use parse-alias on 0'th element, then this possible:
['+', 1, 2, 3, 4] ;-> 10
```

Implementation, conceptually

A command (function) accepts list of CBOR values and returns also a list of them, while being modelled simply like C program's `main()` called from POSIX Shell:

```
int SomeCmdImpl(CBAPT *this,
                int argc,
                cbor_item **argv,  /* input list */
                cbor_item **result /* output, return value tells how many */
)
```

(termin) Why "command" if it's "function"?

Because it enables you to think about subcommands as typical in shell:

```
['git', 'stash', 'apply', args]
['package', 'provide', "SenML", 2.03]
['package', 'require', "DNS", 1.02]
```

Also it's hard to call functions and things like "if" or "foreach" with the same word :)

And last, there is common idiom in Shell (called chain loading or Bernstein chaining) useful for compactness - when one command "prefixes" other:

```
['nice', '-n', 5, 'gcc', "file.c"]
['with-dict', 1, 'process-cbor', {"key1": 6(0)}]
```

Custom functions - DNS example, 1/3

```
10([
  'proc', 'name-comp', ['domain-list'], /procedure with single argument/
  [ /procedure body/
   'set', 'atom-list', [], ';'         /initialize empty array local var/
   'foreach', 'fqdn', '@','domain-list',    /loop variable called "fqdn"/
   [ /outer foreach body - on full domains from 'domain-list' array/
      'set', 'accum', '', ';'                 /empty string to accumulator/
      'foreach', 'label',                     /loop variable called "label"/
          10(['lreverse', '@', 'fqdn']),      /on reversed label list/
      [ /inner foreach body/
        'if', ['is_cbor_uint', '$', 'label'], /reference instead of label?/
        [ /if "then" body/
          'set', 'label', '$',                /to value of CBOR Pointer /
                ['atom-list',                 /in 'atom-list' var: array/
                             '$', 'label']    /index in 'label' in this case/
        ], ';'
        'set', 'accum',              /set accum to string concatenation/
              10(['strconcat',       /of value of "label" variable converted/
                   10(['to_tstr', '$', 'label']),  /to CBOR Major Type 3 /
                   '$', 'accum'      /and previous value of "accum" var  /
                 ]),
        ';',                            /command separator (end of 'set') /
        /now prepend value of "accum" variable to 'atom-list' array/
        'unshift', 'atom-list', '$', 'accum' /TBD is 'linsert' more readable?/
      ],
   ], ';'
   /*** NOTE TODO this lacks error-checking and suppression ***/
   /*** of repeated entries - but you get the core idea     ***/
   'return', 10(['@', 'atom-list'])          /make list from array/
  ], /end of procedure body/
  ';'
  'parse-alias', 'nc', 1, 'name-comp' /for shortened spliced use/
])
```

Custom functions - DNS example, 2/3

Now, assume procedure above is imported by tag or media type, so it can be used in following way:

```
10([simple(0),
    "foo",              /Atom 0/
    'nc',               /call function and substitute it's return values/
    [                   /this will create 7 atoms/
          ["svc", "www", "example", "org"],
          ["datatracker", "ietf", 3]        /atom-list[3] is "org"/
    ],
    "bar",              /Atom 8/
    "baz",              /Atom 9/
    h' /CBAR rump/
      8F                /outer array/
      3D                /Atom 2: CwwwGexampleCorg/
      84                /inner array/
      1E                /Atom 1: CsvcCwwwGexampleCorg /
      5E                /Atom 5: KdatatrackerDietfCorg /
      18 2a             /42/
      1E                /Atom 1: CsvcCwwwGexampleCorg /
      1D                /Atom 0: Cfoo /
      9C                /Atom 8: Cbar /
      9D                /Atom 9: Cbaz /
    '
])
```

Custom functions - DNS example, 3/3

Note how other atoms, except DNS ones, are added anywhere by simply placing function call where you need it, so atoms table now looks like:

```
    / 0      1                     2                  3            /
    ["foo", 'CsvcCwwwGexampleCorg', 'CwwwGexampleCorg', 'GexampleCorg',
    / 4           8      9    /
     'Corg', ..., "bar", "baz"]
```

This gives the following CBOR:

```
[
  "www", "example", "org",
  ["svc", "www", "example", "org"],
  "datatracker", "ietf", "org", 42,
  "svc", "www", "example", "org",
  "foo", "bar", "baz"
]
00000000  8f 43 77 77 77 47 65 78  61 6d 70 6c 65 43 6f 72  |.CwwwGexampleCor|
00000010  67 84 43 73 76 63 43 77  77 77 47 65 78 61 6d 70  |g.CsvcCwwwGexamp|
00000020  6c 65 43 6f 72 67 4b 64  61 74 61 74 72 61 63 6b  |leCorgKdatatrack|
00000030  65 72 44 69 65 74 66 43  6f 72 67 18 2a 43 73 76  |erDietfCorg.*Csv|
00000040  63 43 77 77 77 47 65 78  61 6d 70 6c 65 43 6f 72  |cCwwwGexampleCor|
00000050  67 43 66 6f 6f 43 62 61  72 43 62 61 7a           |gCfooCbarCbaz|
```

Misc

Of course, the definition above is for full implementations - constrained devices will not want to deal with named variables etc. so will hardcode 'name-comp' function in C.

Security

This is not a real programming language (arguably it's not even Turing-complete in formal sense) - not only to be easy to implement, but also for DoS prevention. That is, there are no `goto`-s and "`for`" or "`while`" loops - just a `foreach` on finite-size collection. So execution is guaranteed to terminate in finite time (this is well-known how to do it, see e.g. BPF).

Adding extension functions MUST forbid possibility of infinite loops and also implementations MAY include counters to limit total number of operations.

cbor-packed problems - DNS example from [draft-lenders], 1/5

https://github.com/anr-bmbf-pivot/draft-lenders-dns-cbor/pull/7/files:

Take the following CBOR object o (note that this is intentionally not legal "application/dns+cbor" to illustrate generality).

```
[
  "www", "example", "org",
  ["svc", "www", "example", "org"],
  "org", "example", "org", 42,
  "svc", "www", "example", "org", 42
]
```

cbor-packed problems - DNS example from [draft-lenders], 2/5

This would generate the following virtual table V.

```
[
    ["www", "example", "org"],
    ["example", "org"],
    ["org"],
    ["svc", simple(0)],
    ["org", "example", "org"]
]
```

Not that the sequence "org", "example", "org" is added at index 4 with leading "org", instead of referencing index 2 + index 1 (`simple(2)`, `simple(1)`), as it is its own distinct suffix sequence.

cbor-packed problems - DNS example from [draft-lenders], 3/5

The packed representation of `o` would thus be:

```
TBD28259(
  [
    ["www", "example", "org"],
    ["svc", simple(0)],
    "org", simple(1), 42,
    simple(3), 42
  ]
```

cbor-packed problems - DNS example from [draft-lenders], 4/5

[VG]
Notice here usage of `simple(N)` as an "insertion" of several items (one may say CBOR Sequence).
But `cbor-packed` do NOT support CBOR Sequences! Tables in cbor-packed can contain only complete CBOR items.
Thus, usage of table `V` according to cbor-packed spec will really lead to the following unpacked CBOR:

```
[
  ["www", "example", "org"],
  ["svc", ["www", "example", "org"] ],
  "org", ["example", "org"], 42,
  ["svc", ["www", "example", "org"] ], 42
]
```

which is clearly **not** intended outcome.

cbor-packed problems - DNS example from [draft-lenders], 5/5

So, to support such "array unwrapping" in cbor-packed, one need to effectively DOUBLE number of tags used! But in CBAR/CBAPT approach, this is just additional '@' array unwrapper in the body of procedure which is not transmitted with every packet.

While one can argue that this usage is specific to [dns-cbor] and mandated by Tag 28259, this leads to even worse problem: this is now not EXTENSION of cbor-packed but essentially a FORK of it. Deviation from the spec, and that's while both specs are still at draft stage!

So what will happen when next thing will require serious extension of cbor-packed which claimed table setup tags as main extension point? Combinatorial explosion of specifications interference?..

cbor-packed problems - general

cbor-packed is neither general enough nor efficient by being sharpened to specific task

The central design mistake of cbor-packed is reusing CBOR as-is for everything in it. This can't achieve good compression by definition - you always have tag introducer byte. But even in two byte case, their usage is not effective:

1. cbor-packed allows only 40 values in 2 bytes (1+1 tags)
2. CBAR/CBAPT allows **160** atoms in same 2 bytes - 4x more!

simple values

cbor-packed wants to register 16 simple values - **80%** of 1+0 range - in a way that forbids their usage in any other application for other goals - or otherwise they can't be used together with cbor-packed.

CBAR/CBAPT assigns meaning to `simple(0..19)` values only inside well-defined places - thus, **application protocols can utilize `simple(N)` for their own needs!**

cbor-packed problems - current usage and future extensibility

In it's current form, cbor-packed still has no real-world usage/implementations. As of first months 2025, there are no usage of cbor-packed in IETF drafts beyond simply defining some values for packing tables - a poorly extensile approach by itself, BTW - so these drafts are not tied to cbor-packed and can be painlessly changed to any other similar packing scheme, including CBAR/CBAPT.

The others are [dns-cbor] and drafts extending cbor-packed itself. Problems with DNS were discussed above. Drafts extending cbor-packed itself already cover less cases than (yet evolving) "darft-of-draft" for CBAR/CBAPT.

Reasons to this are fundamental: cbor-packed is done via tags which behaviour is essentially hard-coded to implementations - thus even a **small** change requires separate drafts considering interference of all existing features. CBAPT, on the other hand, is built from small building blocks - functions, like in a programming language - so extending it usually requires **much less effort than set-in-stone tags and their registrations**.

cbor-packed - Early IPv4 allocation mistake

So to conclude: cbor-packed is not worth resources it demands (not only tags, but specifications effort) and can be superseded by CBAR/CBAPT, which gives more promising opportunities to CBOR ecosystem also in other areas.