kaiv for CBOR Users

A Cookbook

The kaiv Project

for kaiv 0.4.0

1 Introduction

You use CBOR — RFC 8949, compact, binary, typed at the byte level, at home in constrained devices and COSE stacks. This cookbook maps CBOR onto **kaiv**: a *text* canonical form with the properties that made you pick a binary one — deterministic encoding, real types (byte strings and datetimes included, which JSON forced you away from), and a validator simple enough for constrained targets: a single forward scan, constant memory, no recursion.

The two are complementary, and conversion runs both ways byte-for-byte: CBOR for the wire, kaiv for everything the wire feeds — diffing, auditing, validating, archiving, and conversion onward to JSON, YAML, XML, or TOML.

Every example is a verified transcript of the reference `kaiv` tool (`cargo install kaiv-cli`); the CBOR inputs are built in-transcript with `printf`, so every byte is on the page.

2 Hello, kaiv

A two-entry CBOR map — `when` carries a tag-0 datetime, `blob` a byte string (`0xCAFE`):

```
$ printf '\xa2\x64when\xc0\x74\x32\x30\x32\x36\x2d\x30\x3
   ↪  7\x2d\x31\x37\x54\x30\x39\x3a\x33\x30\x3a\x30\x30\
   ↪  x5a\x64blob\x42\xca\xfe' > tagged.cbor
$ kaiv import --cbor tagged.cbor
.!kaiv 1
.!types std/enc
```

```
.!types std/time

&datetime
when=2026-07-17T09:30:00Z
&bin
blob=yv4
```

Both of CBOR's beyond-JSON types land on *typed channels*, not lossy approximations: the tag-0 string is typed `&datetime` from `std/time`, and the byte string rides `&bin` from `std/enc` — base64url armor (`yv4` decodes to `CA FE`), declared as binary, never confused with text.

3 The Byte-Identical Round Trip

Typed channels are what make the return trip exact. The same document, back out to CBOR:

```
$ kaiv import --cbor tagged.cbor | kaiv build | kaiv
   ↪  export --cbor | xxd
00000000: a264 7768 656e c074 3230 3236 2d30 372d
   ↪  .dwhen.t2026-07-
00000010: 3137 5430 393a 3330 3a30 305a 6462 6c6f
   ↪  17T09:30:00Zdblo
00000020: 6242 cafe                                bB..
```

The tag re-emits as a tag, the byte string as a byte string — the original bytes, reconstructed. (kaiv is the family's first binary target: `export --cbor` writes raw bytes to stdout.)

4 Non-Finite Floats Survive

CBOR encodes `NaN` and the infinities; JSON cannot. kaiv can — `std/num` names them:

```
$ printf '\xa1\x61x\xf9\x7e\x00' > nan.cbor
$ kaiv import --cbor nan.cbor
.!kaiv 1
.!types std/num

&nan
```

```
x=nan
```

A half-precision `NaN` (`f9 7e00`) became a typed, greppable line. Exporting this document to JSON is a loud error, not a silent `null` — fidelity is never silent in kaiv.

5 Compactness, Measured

kaiv does not compete with CBOR on wire size — it competes on everything after the wire. For a small config document:

```
$ kaiv import config.json | kaiv build > config.daiv
$ kaiv export --json config.daiv | wc -c
94
$ kaiv export --cbor config.daiv | wc -c
63
```

CBOR stays the smallest encoding of the three — and the `.daiv` in the middle is the one you can read, diff, validate in constant memory, and still open in 2040. Use each form where it is strongest; the hub converts between them losslessly.

6 The Mapping

CBOR	kaiv	
map	namespace	
array	`@`-array	
text string	`str` (the default)	
byte string	`&bin` channel	base64url, typed
tag 0 (datetime)	`&datetime`	re-tagged on export
integers (incl. bignum)	`!int`	unbounded decimal
floats (16/32/64-bit)	`!float`	
`NaN` / `Infinity`	`std/num` named types	JSON export refuses
`true`/`false`/`null`	`!bool` / `!null`	
other tags	—	undecoded; see below

7 When to Stay with CBOR

An honest map marks the roads not taken. CBOR remains the right choice when:

- bytes on the wire are the budget — radio links, NB-IoT, flash-constrained logs;
- you live inside the CBOR tag ecosystem — COSE signatures, CWT tokens, and the wider tag registry, of which kaiv decodes tag 0; other tags do not survive the trip;
- you stream indefinite-length items — kaiv documents are complete files, not streams.

kaiv's pull begins where the wire ends: the moment a CBOR payload needs to be read by a person, diffed in review, validated against a contract, or kept for a decade, the canonical text form is the better home — and the round trip back to bytes is exact.

8 Where Next

From here:

- the kaiv specification — the formal grammar and semantics;
- `kaiv help` — the full toolchain surface;
- the sibling cookbooks — kaiv for JSON, YAML, TOML, XML, Protocol Buffers, Avro, ASN.1, and GraphQL users.