CBOR::Core

CBOR Cross-Platform Specification

Anders Rundgren, WebPKI.org

2025-06-21

Why not just continue with JSON forever?

Well…

- IETF standards [in practice] rather use RFC 7493 aka “I-JSON”
- The JavaScript “JSON” object in browsers is extremely limited
- JSON has no “blob” support. Base64 is all over the place
- JSON has no deterministic mode. RFC 8785 is not the same
- JSON has no extension mechanism

Properly packaged, CBOR [RFC 8949] represents a powerful alternative, unhampered by JSON legacy:
https://datatracker.ietf.org/doc/draft-rundgren-cbor-core/

Developers see CBOR as a set of data types + API

CBOR/CDDL	Object Wrapper	JavaScript
`int`	CBOR.Int	`Number`
`bigint`	CBOR.BigInt	`BigInt`
`float`	CBOR.Float	`Number`
`bstr`	CBOR.Bytes	`Uint8Array`
`tstr`	CBOR.String	`String`
`bool`	CBOR.Boolean	`Boolean`
`null`	CBOR.Null	
`[]`	CBOR.Array	
`{}`	CBOR.Map	
`#6.n`	CBOR.Tag	
`#7.n`	CBOR.Simple	`Number`

Non-normative JavaScript Example

Object Wrappers are *bi-directional*, *type-checking*, and *self-rendering*.

IETF

Encode CBOR Data

```
let cbor = CBOR.Map()
    .set(CBOR.Int(1), CBOR.Float(45.7))
    .set(CBOR.Int(2), CBOR.String("Hi there!")).encode();

console.log(CBOR.toHex(cbor));
a201fb4046d9999999999a026948692074686572652 1
```

Decode CBOR Data

```
let map = CBOR.decode(cbor);  // Use result from encode

console.log(map.toString());  // Diagnostic notation
{
  1: 45.7,
  2: "Hi there!"
}

console.log("Value=" + map.get(CBOR.Int(1)).getFloat64());
Value=45.7
```

Non-normative JavaScript Example

Strict Type Checking API – Also in JavaScript

```
console.log("Value=" + map.get(CBOR.Int(1)).getString());
Uncaught CborError: Expected CBOR.String, got: CBOR.Float
```

Decode Diagnostic Notation (Textual CBOR)

```
let cbor = CBOR.diagDecode(`{
# Comments are also permitted
  1: 45.7,
  2: "Hi there!"
}`).encode();

console.log(CBOR.toHex(cbor));
a201fb4046d9999999999a02694869207468657265721
```

Diagnostic Notation as input format has many uses including:

- Config files
- Test data
- Protocol development

Non-normative JavaScript Example

Deterministic Encoding (DE) in CBOR::Core

- DE rationale: *simpler encoder design*, *practically free of cost*, and *potentially improved interoperability*. As a bonus, duplicate key detection becomes default.
- DE primarily relies on *sorted maps*, *normalized integers*, and *normalized floating-point numbers*.
- DE is always "on" for *encoding*.
- DE can optionally be disabled for *decoding* in order to support "legacy" CBOR.
- DE can support novel cryptographic containers, including embedded signatures. This is elaborated on in the draft.
- Application developers should hardly ever have to bother with the inner workings of deterministically encoded CBOR.

What is the alternative to an IETF specified standard?

It is a *potentially fractured* CBOR landscape, where each platform vendor does its own interpretation of what "Useful CBOR" might entail. Presumably this gets a little better than the JavaScript "JSON" object 🤣

For the IETF however, it is now or never!

Unexpected side-effect of a successful IETF standard

Since `CBOR::Core` *targets the "bulk" of SW developers, it could eventually turn out as a de-facto definition of CBOR.*

However, `CBOR::Core` does not in any way modify the CBOR standard; it only provides a *simplified* and *standardized* environment, particularly adapted for more traditional use cases, rather than for resource-constrained embedded systems.

Thanx!