BLOCKCHAIN COMMONS

dCBOR
DETERMINISTIC CBOR

- BINARY
- CONCISE
- SELF-DESCRIBING
- CONSTRAINED ENVIRONMENTS
- PLATFORM/LANGUAGE AGNOSTIC
- STANDARDIZED
- DETERMINISTIC

3. Envelope Format Specification

This section is normative, and specifies the binary format of envelopes in terms of its CBOR components and their sequencing. The formal language used is the Concise Data Definition Language (CDDL) [RFC8610]. To be considered a well-formed envelope, a sequence of bytes MUST be well-formed deterministic CBOR [RFC8949] and MUST conform to the specifications in this section.

- BINARY
- CONCISE
- SELF-DESCRIBING
- CONSTRAINED ENVIRONMENTS
- PLATFORM/LANGUAGE AGNOSTIC
- STANDARDIZED
- DETERMINISTIC

https://datatracker.ietf.org/doc/draft-mcnally-envelope/

THE POINT OF DETERMINISM

- Eliminate choices for how to serialize particular data.
- Where possible, the API enforces these standards.
- Where not possible, developers MUST specify how to serialize and how to validate on deserializing.
- Multiple agents serializing the same data should automatically achieve consensus on the exact form of that data.

WHAT DOES THE SPEC SAY?
RFC-8949

4.2. Deterministically Encoded CBOR

Some protocols may want encoders to only emit CBOR in a particular deterministic format; those protocols might also have the decoders check that their input is in that deterministic format. Those protocols are free to define what they mean by a "deterministic format" and what encoders and decoders are expected to do. This section defines a set of restrictions that can serve as the base of such a deterministic format.

WHAT DOES THE SPEC SAY?
RFC-8949

4.2.1. Core Deterministic Encoding Requirements

- Variable-length integers MUST be as short as possible.
- Floating-point values MUST use the shortest form that preserves the value.
- Indefinite-length arrays and maps MUST NOT be used.
- Map keys MUST be sorted in bytewise lexicographic order of their deterministic encodings.

WHAT DOES THE SPEC SAY?
RFC-8949

4.2.2. Additional Deterministic Encoding Considerations

- Protocols MUST specify the circumstances under which a data item MUST or MUST NOT be tagged.
- Protocols allowing the use of BigNums ≥ 2^{64} (tags 2 and 3) MUST specify whether values $<2^{64}$ MUST use regular integer encodings.
- Protocols allowing the use of floating-point numbers must decide how to encode values like -0.0, NaN/Signalling NaN, subnormal values, etc.

WHAT DOES BLOCKCHAIN COMMONS SAY?

- Deterministic encoding is essential for cryptographic “smart documents” like Gordian Envelope.
- All of our existing CBOR specs are already deterministic encoding-compliant.
- Being opinionated is good.
- Enforcing opinionated best practices at the software/API level is even better.

HOW MANY EXISTING CBOR IMPLEMENTATIONS DIRECTLY SUPPORT DETERMINISTIC ENCODING AS A CORE VALUE?

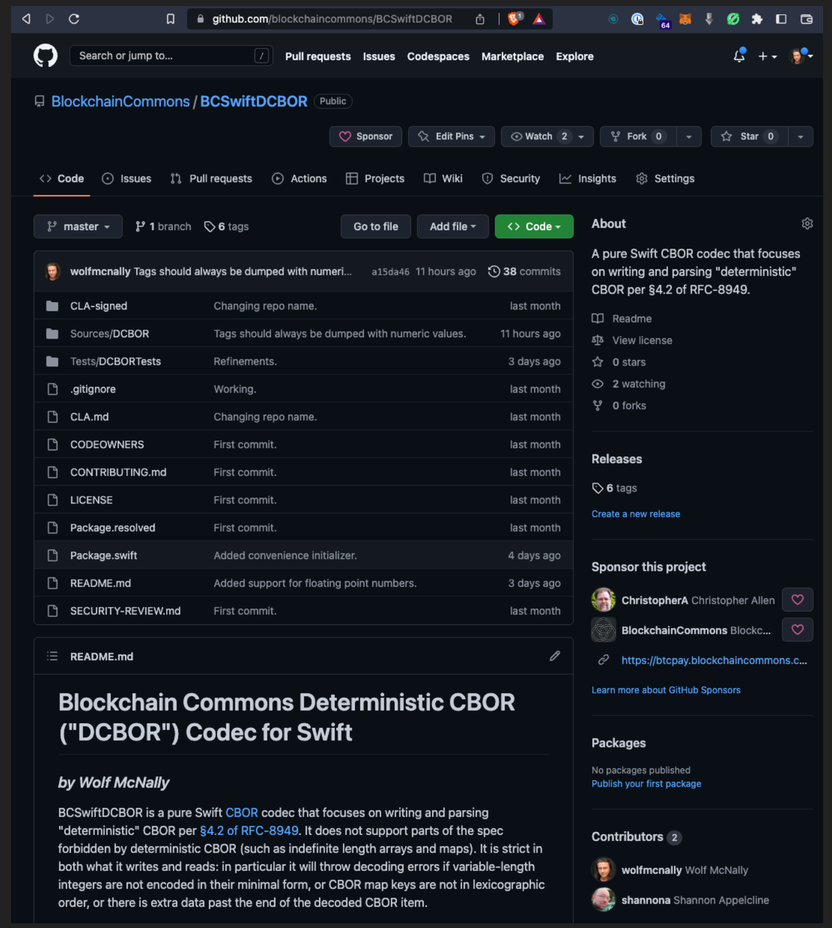
404

(none found)

SO NOW WE'VE BUILT TWO OF THEM...

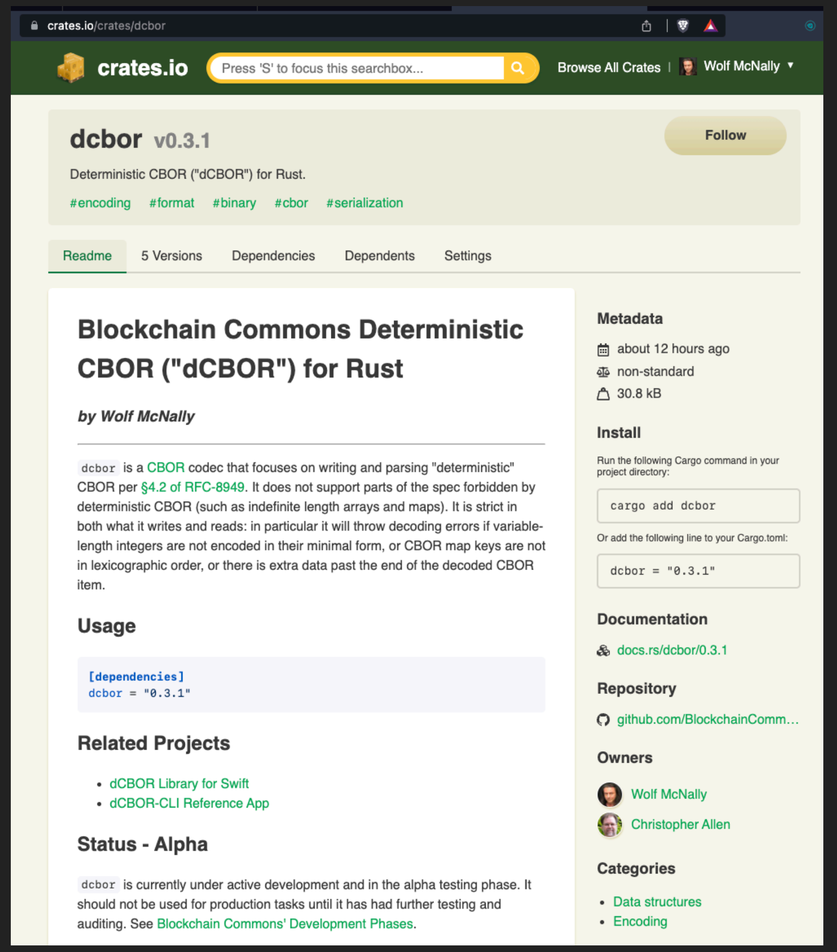
dCBOR Swift

https://github.com/BlockchainCommons/BCSwiftDCBOR



dCBOR Rust

https://crates.io/crates/dcbor



GOALS FOR dCBOR

- Make it easy to write and read deterministic CBOR (dCBOR) that complies with RFC-8949 §4.2.1. Core Deterministic Encoding Requirements.
- Be strict about what is written and read.
  - Make it hard to write non-compliant dCBOR.
  - Make it an error to read non-compliant dCBOR.
- Facilitate as much as possible the considerations in RFC-8949 §4.2.2. Additional Deterministic Encoding Considerations.

Provide protocols (Swift) and traits (Rust) to make structures CBOR-friendly.

- `CBORCodable` conformance adds serialization/deserialization to any type.
  - Many fundamental built-in types conform including integers, floating point values, strings, byte strings, arrays, booleans, and dates.
- `CBORTaggedCodable` adds a tag that is always written on serialization and expected on deserialization.
- No attempt has been made to make dCBOR compatible with either the `Codable` protocol (Swift) or the `SerDe` serialization framework (Rust).
  - A lot of work for little benefit, with many sharp edge/corner cases to deal with.
  - If this is something you desire, we welcome PRs!

Encoding of Numeric Values

- All encoded numeric values use the shortest possible serialization.
  - Integers are 8, 16, 32, or 64 bits.
  - Floating point values 16, 32, or 64 bits.
- Floating point values with no fractional part are serialized as integers if possible.
  - This means that 0.0, -0.0, and 0 are all serialized exactly the same way.
- After deserialization, any numeric value can be extracted as a floating point value.
- Attempting to extract an integer from a numeric value with a fractional part is an error.
- Attempting to deserialize a dCBOR stream with any numeric values not in their canonical shortest form is an error.
- No attempt is currently made to canonicalize things like NaN/Signalling Nan, or subnormal values.

Encoding of Maps

- RFC-8949: Map keys MUST be sorted in bytewise lexicographic order of their deterministic encodings.
- dCBOR libraries provide a special-purpose `Map` structure that keeps key-value pairs in canonical sorted order as they are inserted or removed.
- Provides iteration through key-value pairs in canonical order.
- In some other ways they behave like a normal dictionary/map, but they are primarily intended for use during the serialization/deserialization process.
- Deserialization of out-of-order map keys is an error.

Output of CBOR diagnostic notation and annotated hex dumps

- The deserialized `CBOR` type has `diagnostic()` and `hex()` methods.
- Can be provided with `knownTags` argument that provides names for tags.

```
304(   ; crypto-keypath
   {
      1:
      [23, false, 23, true, 33, false]
   }
)
```

```
d9 0130        # tag(304)   ; crypto-keypath
   a1          # map(1)
      01       # unsigned(1)
      86       # array(6)
         17    # unsigned(23)
         f4    # false
         17    # unsigned(23)
         f5    # true
         1821  # unsigned(33)
         f4    # false
```

Validations performed while decoding or extracting

```swift
/// An error encountered while decoding CBOR.
public enum CBORDecodingError: LocalizedError, Equatable {
    /// Early end of data.
    case underrun

    /// Unsupported value in CBOR header.
    ///
    /// The case includes the encountered header as associated data.
    case badHeaderValue(encountered: UInt8)

    /// An integer was encoded in non-canonical form.
    case nonCanonicalInt

    /// A floating point value was encoded in non-canonical form.
    case nonCanonicalFloat

    /// An invalidly-encoded UTF-8 string was encountered.
    case invalidString

    /// The decoded CBOR had extra data at the end.
    ///
    /// The case includes the number of unused bytes as associated data.
    case unusedData(Int)

    /// The decoded CBOR map has keys that are not in canonical order.
    case misorderedMapKey

    /// The decoded CBOR map has a duplicate key.
    case duplicateMapKey

    /// The decoded integer could not be represented in the specified integer type.
    case integerOutOfRange

    /// The decoded value was not the expected type.
    case wrongType

    /// The decoded value did not have the expected tag.
    ///
    /// The case includes the expected tag and encountered tag as associated data.
    case wrongTag(expected: Tag, encountered: Tag)

    /// Invalid CBOR format. Frequently thrown by libraries depending on this one.
    case invalidFormat
}
```

CHRISTOPHER ALLEN
christophera@lifewithalacrity.com
@BlockchainComns

WOLF MCNALLY
wolf@wolfmcnally.com
@WolfMcNally