# CBOR data definition language

## draft-greevenbosch-appsawg-cbor-cddl-01

Bert Greevenbosch

# Summary

- RFC 7049 defines the Concise Binary Object Representation (CBOR) data format.
- The RFC defines a representation of data in a structured way.
- The document does not define a notation writing down the structure of CBOR data.
- draft-greevenbosch-appsawg-cbor-cddl provides such notation.

# Why is there a need for this?

- **To have a unified way to define CBOR data structures while writing standards.**
  - Right now, such structures are either described in text, or need to be seen from a data instance.
    - CBOR's diagnostic notation
    - Binary dump
  - A unified notation allows quicker understanding of CBOR data structures across different documents.

# Example 1

## CDDL

```
*PointOfInterest {
  name            : tstr;
  description     : tstr;
  ?coordinates    : Coordinates;
  comments        : map( tstr, tstr );
}

# Coordinates may be omitted when unknown
*Coordinates {
  latitude        : int; # multiplied by 10^5
  longitude       : int; # multiplied by 10^5
}
```

## Instance Binary Dump

***DIAGNOSTIC:***

```
[_ "Charles Dickens Museum",
   "Former residence of the writer",
   [ 5152392, -11690 ],
   {_ "Alice": "...", "Bob": "..." }
]
```

***BINARY:***

```
9F             # indefinite length array PointOfInterest
  76 ...       # "Charles Dickens Museum"
  78 1E ...    # "Former residence of the writer"
  82           # coordinates as 2 element array
     1A 00 4E 9E 88 # longitude 51.52392 N(+)
     39 2D A9       # latitude 0.11690 W(-)
  BF           # infinite length map "comments"
     65 ...    # "Alice"
     78 3A ... # a 58 characters comment
     63 ...    # "Bob"
     78 45 ... # a 69 characters comment
  FF           # end of "comments" map
FF             # end of "PointOfInterest"
```

# Example 2

## CDDL

```
PointOfInterest: map( tstr ) {
  "name"          : tstr;
  "description"   : tstr;
  ?"coordinates"  : Coordinates;
  "comments"      : map( tstr, tstr );
}

# Coordinates may be omitted when unknown
*Coordinates {
  latitude        : int; # multiplied by 10^5
  longitude       : int; # multiplied by 10^5
}
```

## Instance

***DIAGNOSTIC:***

```
{_ "name": "Charles Dickens Museum",
   "description": "Former residence of the writer",
   "coordinates": [ 5152392, -11690 ],
   "comments": {_ "Alice": "...", "Bob": "..." }
}
```

***BINARY:***

```
BF             # indefinite length map PointOfInterest
  64 ...       # "name"
  76 ...       # "Charles Dickens Museum"
  6B ...       # "description"
  78 1E ...    # "Former residence of the writer"
  6B ...       # "coordinates"
  82           # coordinates as 2 element array
     1A 00 4E 9E 88 # longitude 51.52392 N(+)
     39 2D A9  # latitude 0.11690 W(-)
  68 ...       # "comments"
  BF           # infinite length map "comments"
     65 ...    # "Alice"
     78 3A ... # a 58 characters comment
     63 ...    # "Bob"
     78 45 ... # a 69 characters comment
  FF           # end of "comments" map
FF             # end of "PointOfInterest" map
```

# Features of CDDL

- Clear signalling of the structure of the data.
- Support for basic data types.
  - E.g. uint (unsigned int), tstr (text string), bool
- Support for maps and arrays.
  - Optional signalling of pre-defined keys for maps.
- Support for tags.
  - Pre-defined abbreviations for common tags, such as bignum, bigfloat and uri.

# Open Issues

- Machine parsing as a design criteria?
  - Helps keeping the language unambiguous.
  - Forces a clear syntax.
  - Could be used for automatic processing.
    - An attractive nuisance?
- Optional variables
  - Could be useful in real-world scenarios.
  - Hard to express their existence.
  - Due to possible ambiguity, in conflict with the machine parsing criterion.
- Input is welcome!

# Thank you for your attention!