sid-extension & pycoreconf

Javier Fernandez
Laurent Toutain

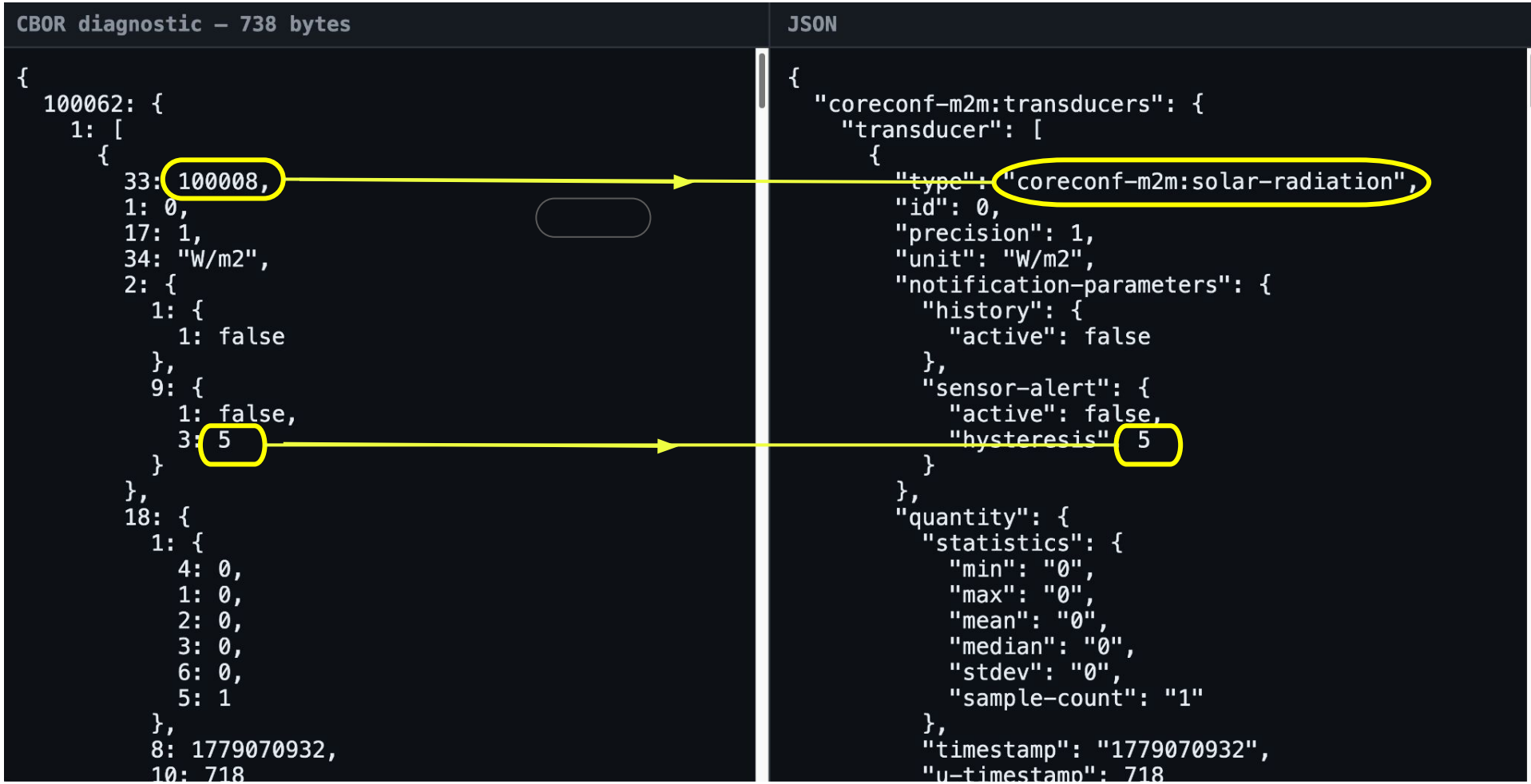
types

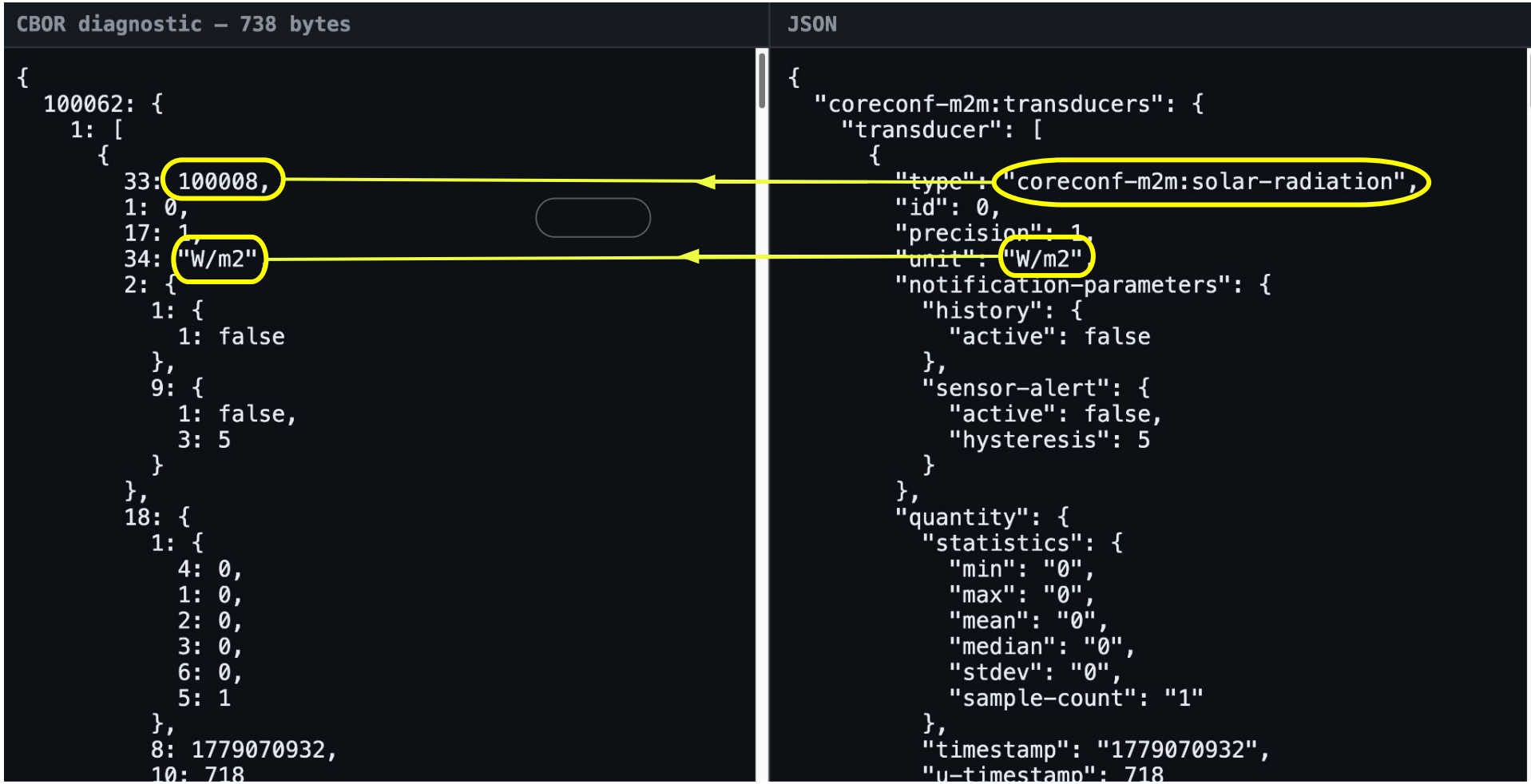
CBOR diagnostic — 738 bytes

```
{
  100062: {
    1: [
      {
        33: 100008,
        1: 0,
        17: 1,
        34: "W/m2",
        2: {
          1: {
            1: false
          },
          9: {
            1: false,
            3: 5
          }
        },
        18: {
          1: {
            4: 0,
            1: 0,
            2: 0,
            3: 0,
            6: 0,
            5: 1
          },
          8: 1779070932,
          10: 718
```

JSON

```
{
  "coreconf-m2m:transducers": {
    "transducer": [
      {
        "type": "coreconf-m2m:solar-radiation",
        "id": 0,
        "precision": 1,
        "unit": "W/m2",
        "notification-parameters": {
          "history": {
            "active": false
          },
          "sensor-alert": {
            "active": false,
            "hysteresis": 5
          }
        },
        "quantity": {
          "statistics": {
            "min": "0",
            "max": "0",
            "mean": "0",
            "median": "0",
            "stdev": "0",
            "sample-count": "1"
          },
          "timestamp": "1779070932",
          "u-timestamp": 718
```


CBOR diagnostic — 738 bytes
JSON
{
100062: {
1: [
{
33: 100008,
1: 0,
17: 1,
34: "W/m2",
2: {
1: {
1: false
},
9: {
1: false,
3: 5
}
},
18: {
1: {
4: 0,
1: 0,
2: 0,
3: 0,
6: 0,
5: 1
},
8: 1779070932,
10: 718
{
"coreconf-m2m:transducers": {
"transducer": [
{
"type": "coreconf-m2m:solar-radiation",
"id": 0,
"precision": 1,
"unit": "W/m2",
"notification-parameters": {
"history": {
"active": false
},
"sensor-alert": {
"active": false,
"hysteresis": 5
}
},
"quantity": {
"statistics": {
"min": "0",
"max": "0",
"mean": "0",
"median": "0",
"stdev": "0",
"sample-count": "1"
},
"timestamp": "1779070932",
"u-timestamp": 718


CBOR diagnostic — 738 bytes
JSON
{
100062: {
1: [
{
33: 100008,
1: 0,
17: 1,
34: "W/m2"
2: {
1: {
1: false
},
9: {
1: false,
3: 5
}
},
18: {
1: {
4: 0,
1: 0,
2: 0,
3: 0,
6: 0,
5: 1
},
8: 1779070932,
10: 718
{
"coreconf-m2m:transducers": {
"transducer": [
{
"type": "coreconf-m2m:solar-radiation",
"id": 0,
"precision": 1,
"unit": "W/m2",
"notification-parameters": {
"history": {
"active": false
},
"sensor-alert": {
"active": false,
"hysteresis": 5
}
},
"quantity": {
"statistics": {
"min": "0",
"max": "0",
"mean": "0",
"median": "0",
"stdev": "0",
"sample-count": "1"
},
"timestamp": "1779070932",
"u-timestamp": 718

Type

- CBOR structure = JSON structure
  - key: delta encoded or string following the hierarchy
  - value type is needed for leaf value conversion
- Include type into the sid file
  - convenient since sid ⇔ YANG name is present
- 4 types
  - YANG basic types (uint, int, float, string, ….)
  - identityref
  - enumeration
  - bits
- union

test sid-extension in pyang

```
%uvx --from git+https://github.com/ltn22/pyang@sid-extension
pyang -p ../pyang/modules/ietf/ --sid-generate-file=100000:400
--sid-extension coreconf-m2m@2026-03-29.yang
WARNING: Module 'ietf-geo-location' imported without revision,
using latest revision 2022-02-11
File coreconf-m2m@2026-03-29.sid created
Number of SIDs available : 400
Number of SIDs used : 98
```

coreconf-m2m@2026-03-29.sid


```
{
   "namespace": "data",
   "identifier": "/coreconf-m2m:characteristics",
   "status": "unstable",
   "sid": "100018"
},

{
   "namespace": "data",
   "identifier": "/coreconf-m2m:history/time-series/id",
   "status": "unstable",
   "sid": "100045",
   "type": "uint8"
},

{
   "namespace": "data",
   "identifier": "/coreconf-m2m:sensor-alert/target/type",
   "status": "unstable",
   "sid": "100058",
   "type": "identityref"
 },

{
   "namespace": "data",
   "identifier": "/coreconf-m2m:transd
   "status": "unstable",
   "sid": "100090",
   "type": {
     "0": "source",
     "1": "receiver"
    }
},
```

Example

```
sid_path = "coreconf-m2m@2026-03-29.sid"
ccm = pycoreconf.CORECONFModel(sid_path)

cbor_data = ccm.encode_json(json_file)
decoded_json = ccm.decode(cbor_data)
```

see:
https://github.com/alex-fddz/pycoreconf/blob/main/samples/datastore/main.py

–sid-extension next steps

- Does not handle bits yet
  - hex string? (experimental)
- .sid is a JSON file from a YANG Data Model
- augment current format.

- see https://github.com/ietf-wg-schc/YANG-DM/blob/management/ietf-sid-file-extension%402026-04-28.yang

```
structure sid-file:
  ..
   +-- item* [namespace identifier]
      +-- status?          enumeration
      +-- namespace        enumeration
      +-- identifier       union
      +-- sid              sid
      +-- sid-ext:type
         +-- sid-ext:base-type?      yang-type
         +-- sid-ext:identityref?    uint64
         +-- sid-ext:enumerate
         |  +-- sid-ext:enum-value?    uint32
         |  +-- sid-ext:enum-name?     string
         +-- sid-ext:bits
            +-- sid-ext:bit-position?    uint32
            +-- sid-ext:bit-name?        string
```

key_mapping

key_mapping

Used to navigate inside the data structure:

- When reaching a list, the leaf acting as a key must be known.
- `key_mapping` in the sid file adds this information:
  - JSON key indicates which nodes are YANG list.
  - JSON array gives the list of YANG nodes acting as a YANG key.
  - association the value in this YANG key with the values given in the request.

```
"key-mapping": {
    "100063": [
      100096,
      100064
    ],
    "100044": [
      100050,
      100045
    ],
    "100056": [
      100058,
      100057
    ]
}
```

pycoreconf datastore, direct code manipulation:

```
ds = ccm.create_datastore_from_cbor(cbor_data)
transducer_keys = ds["/transducers/transducer"]
xpath_entry =
 "/transducers/transducer[type='coreconf-m2m:solar-radiation'][id='0']"
entry = ds[xpath_entry]
ds[xpath_entry+"quantity/statitics/sample_count"] += 1

new_xpath =
 "/transducers/transducer[type='coreconf-m2m:solar-radiation'][id='1']"
ds[new_xpath] = {'precision': 3}

for pred in ds.predicates("/transducers/transducer"):
   xpath_sc = f"/transducers/transducer{pred}/quantity/…/sample-count"
   ds[xpath_sc] += 1
```

Interaction with CoAP

```
async def render_fetch(self, request):

    if not request.payload:
        return Message(payload=self.db.to_coreconf_cbor(), content_format=142
    item = request.payload

    if type(item) is int:
        leaf_sid = item
        keys = []
    else:
        leaf_sid, *keys = item

    raw = cbor.loads(self.db.to_coreconf_cbor())
    result = self.db.model.findSID(raw, sid=leaf_sid, keys=keys, depth=depth)
    if result is not None:
        return Message(payload=result.getvalue(), content_format=142)
```

Interaction with CoAP

```
async def render_ipatch(self, request):

    for key, value in cbor.loads(request.payload).items():
        xpath = f"/{MODULE}:" + self.db.datastore._create_xpath(key[0], key[1:]).lstrip("/")
        incoming = next(iter(json.loads(self.db.model.toJSON(cbor.dumps(value))).values()))
        existing = self.db.datastore[xpath]
        merged = {**existing, **incoming}

        self.db.datastore[xpath] = merged

    return Message(code=numbers.codes.Code.CHANGED)
```