

# SenML Data Value Content- Format Indication

draft-ietf-core-senml-data-ct-01

Ari Keränen, Carsten Bormann

IETF 107+, 2020-04-29, in the cloud

# Examples

```
{"n": "nfc-reader", "vd": "gmNmb28YKg", "ct": "60" }
```

```
{"n": "nfc-reader-42",  
  "vd": "H4sIAA+dmFwAAzMx0jEZMAQALnH8Yn0AAAA",  
  "ct": "text/csv@gzip" }
```

# Feature objective: extensibility

- `ct` is generally ignorable (like any new SenML field)
- But we would like to also have a “must understand” version, `ct_`
- Issue: Interaction between the two (`bct`, `bct_`) and resolved records
- Would prefer to have specific information (in record) override base
- But now, that happens only separately, within the thread for each field name!

# RFC 8428: “Must understand” and “\_”?

- »Extensions that are **mandatory** to understand to correctly process the Pack MUST have a label name that ends with the “\_” character.«
- »Applications MUST ignore any JSON **key-value pairs** that they do not understand unless the key ends with the “\_” character, in which case an error MUST be generated.«  
(12.3.1 for senml+json, equivalent text for other representations)
- So a receiver is free to ignore a **key-value combination** if it doesn’t understand the key **or** if it doesn’t understand the combination
- Note that foo and foo\_ are different fields from a SenML perspective, except possibly by their semantic definition
  - convention: don’t define a foo and a foo\_ that are unrelated

# RFC 8428: ct, ct\_, bct, bct\_

- Resolving algorithm can be performed without understanding field semantics: no inter-field interaction
  - Fields do define how base value and given value for that field mix
  - »A future specification that defines new base fields needs to specify how the field is resolved.«
- Resolving is not influenced by unrelated fields (ct vs. ct\_): It happens **separately** for ct and for ct\_
- The rules applying to a record are applied **after** resolving
- But we need to look at examples having some of these four and see whether what we built makes sense

# Solution option #1

- Do not apply base value (bct or bct\_) if a current value (ct or ct\_) exists in the record
- Not supported by RFC 8428
  - Would require using new version/feature for SenML

# Solution option #2

- Future specification need to specify semantics of the "safe-to-ignore" and "must understand" versions of the same field in the same record
  - ct\_ is the first registration of "must understand" fields
  - Can be handled as DE guidance and clarified in SenML-bis?
- Easy to avoid problem: don't mix the two variants in the Packs
  - but also need to enable combining packs easily
- For ct draft: if both exist in the same Record: ct\_ overrides ct (i.e., ignore/remove "safe-to-ignore" version)
- Not perfect, but we don't know better without new SenML version

# What we don't like about solution #2

- If a pack has a `bct_`, you can no longer usefully use `bct` or `ct` *from that position on*
- That is a limitation, but it doesn't detract from other useful combinations
- Workaround: Instead of using `bct_`, use `ct_` once to check the must-understand feature; can use `bct` then
- To do: designated expert to write a wiki page explaining all this



# Backup

# Mixing b and \_ fields: what are the resolution rules?

1) [ {"bfoo\_":42, "n":"t1", "v":1}, {"n":"t2", "v":2}, {"foo": 1, "n":"t3", "v":3} ]

2) [ {"bfoo\_":42, "n":"t1", "v":1}, {"n":"t2", "v":2}, {"foo\_": 1, "n":"t3", "v":3} ]

3) [ {"bfoo":42, "n":"t1", "v":1}, {"n":"t2", "v":2}, {"foo\_": 1, "n":"t3", "v":3} ]

4) [ {"bfoo":42, "n":"t1", "v":1}, {"n":"t2", "v":2}, {"foo": 1, "n":"t3", "v":3} ]

# Mixing b and \_ fields: resolved

1)

```
[  
  {"bfoo_":42, "n":"t1", "v":1},  
  {"n":"t2", "v":2},  
  {"foo": 1, "n":"t3", "v":3}  
]
```

```
[  
  {"foo_":42, "n":"t1", "v":1},  
  {"foo_":42, "n":"t2", "v":2},  
  {"foo": 1, "foo_":42, "n":"t3", "v":3}  
]
```

# Mixing b and \_ fields: resolved

2)

```
[  
  {"bfoo_": 42, "n": "t1", "v": 1},  
  {"n": "t2", "v": 2},  
  {"foo_": 1, "n": "t3", "v": 3}  
]
```

```
[  
  {"foo_": 42, "n": "t1", "v": 1},  
  {"foo_": 42, "n": "t2", "v": 2},  
  {"foo_": 1, "n": "t3", "v": 3}  
]
```

# Mixing b and \_ fields: resolved

3)

```
[  
  {"bfoo":42, "n":"t1", "v":1},  
  {  
    "n":"t2", "v":2},  
  {"foo_": 1, "n":"t3", "v":3}  
]
```

```
[  
  {"foo":42, "n":"t1", "v":1},  
  {"foo":42, "n":"t2", "v":2},  
  {"foo_": 1, "foo":42, "n":"t3", "v":3}  
]
```

# Mixing b and \_ fields: resolved

4)

```
[  
  {"bfoo": 42, "n": "t1", "v": 1},  
  {  
    "n": "t2", "v": 2},  
  {"foo": 1, "n": "t3", "v": 3}  
]
```

```
[  
  {"foo": 42, "n": "t1", "v": 1},  
  {"foo": 42, "n": "t2", "v": 2},  
  {"foo": 1, "n": "t3", "v": 3}  
]
```