Objenious
Codec definition

Version 2.0.0

Objenious
by Bouygues Telecom

1 Introduction

This document describes the configuration of codecs on the Objenious platform.

Codecs decode binary payloads to JSON objects. 3 types of codecs are currently implemented:

- Generic codec,
- NKE Batch, for NKE devices using batch mode,
- Senlab, for Sensing Lab devices using batch mode.

2 Generic codec

2.1 Syntax

The codec definition has three sections:

- "defaults": this section defines a list of global settings,
- "attributes": this section lists all possible attributes and their type,
- "format": this section defines the way the attributes are laid out.

Example codec:

```
{
    "defaults": {
       "endian": "little"
    },
    "attributes": {
      "id": {
          "type": "int",
          "hidden":true,
          "length": 8
      },
      "battery_level": {
          "type": "uint",
          "unit": 255,
          "length": 8
      },
      "internal": {
          "type": "int",
          "hidden":true,
          "length": -16
      },
      "temperature": {
          "type": "int",
          "length": 16,
          "divide": 16
      }
    },
    "format": [
      {
          "attributes": ["id"]
      },
```

```
      {
        "if": "id == 1",
        "then": [
            {
                "attributes": ["battery_level", "internal", "temperature"]
            }
        ]
      }
    ]
}
```

2.2 Defaults

The defaults section lists the default settings to be used for all attributes:

- "endian": order of bytes (https://en.wikipedia.org/wiki/Endianness)

big	big endian (default)	`0xaabbccdd`
little	little endian with 1-byte atomic elements	`0xddccbbaa`
little2	little endian with 2-bytes atomic elements	`0xccddaabb`
big2_swap	big endian with 2-bytes atomic elements, swapped (middle endian)	`0xbbaaddcc`

- "order": order of bits - "msb" (default) or "lsb",

msb	Most Significant Bit first (default)	`7 = 0b00000111`
lsb	Least Significant Bit first	`7 = 0b11100000`

- "negative": representation of negative numbers (https://en.wikipedia.org/wiki/Signed_number_representations)

2_complement	2 complement (default)	`-7 = 0b11111001`
sign_magnitude	Sign Magnitude (1 bit for the sign + N bits for the absolute value)	`-7 = 0b10000111`

Those defaults can be overridden in attributes.

2.3 Attributes

An attribute has the following properties:

- "type": an attribute can have the following types

int	Signed integer (big endian, MSB, 2 complement)
uint	Unsigned integer
float	IEEE 754 floating point number
bool	1-bit boolean value, true = 1/false = 0
char	ASCII 7 bits string, fixed length
string	ASCII 7 bits string, variable length (0 terminated)
binary	Binary data, stored in a hexadecimal string
timestamp	Absolute timestamp of the measurements – in seconds since 1/1/1970 00:00:00 UTC (unix epoch time)
reltimestamp	Age of the measurement relative to the message timestamp – in

	seconds

- "length": number of bits,
- "variable": the attribute has a variable length, the first "length" bits found define the number of bytes of the attribute,
- "multiply": the decoded value will be multiplied by the value to get the final value,
- "divide": the decoded value will be divided by the value to get the final value,
- "hidden": if set to "true", the attribute will not be part of the decoded object,
- "endian": (see before)
- "order": (see before),
- "negative": (see before),
- "attributes": the attribute includes a list of other attributes.

Example 1: A 16 bits integer, 2-complement negative numbers. A "250" value will be decoded as "15.625" (250/16).

```
"temperature": {
   "type": "int",
   "length": 16,
   "divide": 16
}
```

Example 2: A 32 bits integer including other attributes – endianness will be applied on the container attribute before parsing attribute1 and attribute2.

```
 "container": {
   "type": "uint",
   "hidden": true,
   "length": 32,
   "attributes": ["attribute1", "attribute2"]
},
"attribute1": {
   "type": "uint",
   "length": 4
},
"attribute2": {
   "type": "uint",
   "length": 3
}
```

Example 3: A string of variable length: the first byte contains the number of chars of the following string.

```
 "string": {
   "type": "char",
   "length": 8,
   "variable": true
}
```

Example 4: A timestamp (absolute). The attribute will be hidden (absent from decoded payload) but the measurement will be recorded at the decoded timestamp.

```
 "timestamp": {
   "type": "timestamp",
   "length": 32
}
```

2.4 Format

Format is defined as a list of parts. A part can either be:

- "attributes",
- "stop",
- "abort",
- "copy",
- "rename",
- "delete",
- "eval",
- "decode",
- "if/then",
- "repeat".

2.4.1 attributes

Syntax: {"attributes": ["attribute", "attribute"]}

Decode a list of attributes from the payload. Decoding will stop if the end of the buffer is reached

Example 1: a payload with a single attribute

```
{
    "attributes": ["temperature"]
}
```

Example 2: a payload with a specific timestamp (timestamp is defined as a "timestamp" attribute)

```
{
    "attributes": ["timestamp", "temperature"]
}
```

Example 2: a payload with two measurements at different timestamps (timestamp is defined as a "timestamp" attribute). A "attributes" part should not contain more than one timestamp attribute.

```
{
    "attributes": ["timestamp", "temperature"]
},
{
    "attributes": ["timestamp", "temperature"]
}
```

2.4.2 stop

Syntax: {"stop": true}

Stop decoding, previously decoded attributes will be returned.

2.4.3 abort

Syntax: {"stop": true}

Stop decoding, no attributes will be returned.

2.4.4 copy

Syntax: {"copy": ["from", "to"]}

Copy the value of an attribute to another attribute.

2.4.5 rename

Syntax: {"rename": ["from", "to"]}
Rename an attribute.

2.4.6 delete

Syntax: {"delete": ["attribute", "attribute"]}
Delete multiple attributes.

2.4.7 eval

Syntax: {"eval": ["attribute", "expression"]}
Compute a new attribute, based on other attributes.

Example:

```
{
    "eval": ["power", "voltage * current"]
}
```

2.4.8 decode

Syntax: {"decode": ["binary attribute", "attribute", "attribute"]}
Decode a previously found binary attribute and decode new attributes based on it.

Example:

```
{
    "decode": ["buffer", "voltage", "current"]
}
```

If buffer is 0xfa01 and voltage/current are both 8-bit unsigned ints, voltage will decode to 250 and current to 1.
Note: the buffer attribute must have been defined with the "binary" type, otherwise an error will be returned.

2.4.9 if/then

Syntax {"if": "conditions", "then": [part, part]}
Test the conditions and execute the specified parts if the condition is true.

- Conditions use the following operators: "==" (equals to), "!=" (different from), ">", ">=" (greater than), "<", "<=" (less than), "&&" (logical and) and "||" (logical or). Spaces have to be present before/after the operator.
- Condition can test:
  - previous values (hidden or not),
  - protocol values (e.g. port for LoRa devices)

Example 1: temperature_present is defined as a boolean

```
{
    "if": "temperature_present",
    "then": [
        {
            "attributes": ["Temperature"]
        }
    ]
}
```

Example 2: Complex conditions

```
{
  "if": "command_id == 1 && cluster_id == 1026 && attribute_id == 1",
    "then": [
        {
            "attributes": ["TemperatureMin", "TemperatureMax"]
        }
    ]
}
```

Example 3: stop after having decoded an attribute

```
{
  "if": "command_id == 1 && cluster_id == 1026 && attribute_id == 1",
    "then": [
        {
            "attributes": ["TemperatureMin"]
        },
        {
            "stop": true
        }
    ]
}
```

Example 4: aborting

```
{
  "if": "command_id == 1 && cluster_id != 1026",
    "then": [
        {
            "abort": true
        }
    ]
}
```

2.4.1 repeat

Syntax: {"repeat": [part, part]}

Decode the specified attributes from the payload. Decoding will stop if the end of the buffer is reached

Example 1: a dynamic list of attributes

```
{
  "repeat": [
    { "attributes":["type"] },
    {
      "if": "type == 0", "then": [ { "attributes":["state"] } ]
    },
    {
      "if": "type == 1", "then": [ { "attributes":["temperature"] } ]
    }
 }
```

Example 2: a list of measurements at different timestamps

```
  "repeat": [
```

```
        { "attributes":["timestamp", "temperature"] }
    }
```

3 NKE Batch

Using batch mode, NKE devices can report multiple data points, either from a single attribute or from multiple attributes, at multiple times.

The codec needs to be configured in a similar way as the *br_uncompress* tool provided by NKE.

The configuration has 2 properties:

- "tag_size": the size of tags (e.g. 1)
- "measures": the list of attributes. Each attribute has the following properties:
  - "attribute": the name of the attribute,
  - "type": the type of the values (e.g. 7),
  - "resolution": the resolution of the measurement increments,
  - "divide": (see before).

Example:

```
{
    "tag_size": 1,
    "measures": [
        {
            "attribute": "temperature",
            "type": 7,
            "resolution": 100,
            "divide": 100
        }
    ]
}
```

4 Senlab

Using batch mode, SensingLab devices can report multiple data points. This codec only decodes complex payloads reporting multiple data points. Single payloads are decoded using a generic decoder configuration.

The configuration has 2 properties:

- "device_type", e.g. SenlabM, SenlabT, SenlabH...
- "device_version", e.g. 1.1

Example:

```
{
    "device_type": "SenlabT",
    "device_version": "1.1"
}
```

5 Changes

5.1 Version 1.1.0

- Add "defaults" section, remove "transform"
- Use "2 complement" instead of "sign-magnitude" as default
- Rename "unit" to "divide" and add "multiply"

5.2 Version 1.1.1

- Rename the incorrect attribute name “endianness” to the correct “endian”

5.3 Version 1.2.0/1.2.1/1.2.2

- Add “stop” and “abort”

5.4 Version 1.3.0

- Add “resolution” to NKE Batch decoder

5.5 Version 1.4.0/1.4.1

- Add “little2” and “big2_swap” endianness
- Add “binary” type
- Add “copy”, “rename”, “delete”, “eval” and “decode” operators

5.6 Version 2.0.0

- Add “string”, “timestamp” and “reltimestamp” types
- Add “repeat” operator