Guidebook of Machine Readable APPENDIX (MRA)

Revision history

Date	Version	Description
2021.12.01	1.0.0	1st release
2022.05.27	1.1.0	2nd release changed number_0-100% to number_0-100percent

1. Abstract

In order to make it easy to utilize the contents of "APPENDIX Detailed Requirements for ECHONET Device objects " for the software development, we provide the information described in this APPENDIX as JSON format data files. These files are called as "Machine Readable APPENDIX (MRA)". This document describes the data format of MRA.

MRA is for ECHONET Lite only though the APPENDIX describes the properties of both ECHONET Lite and ECHONET (which properties describe GetM and SetM in the access rule).

APPENDIX has been revised with the release version name "A", "B", "C" and so on. Applications and services that utilize ECHONET Lite must support not only the latest version products but also the previous version products in the market and at home. Considering this situation, MRA covers the previous release version too.

2. Version number

The format version and data version are defined in the MRA version. They are described in the meta data later. The format version is synchronized with the version of this document. The data version is used to distinguish the data to be published. The version number is described as follows.

Data format

> xxx.yyy.zzz
>
> xxx: major structure change (without format compatibility)
> yyy: minor structure change (with format compatibility)
> zzz: fix of typo

Data

> xxx.yyy.zzz
>
> xxx: data with major structure change (without format compatibility)
> yyy: addtion of devices, data updates by the APPENDIX release updates
> zzz: fix of typo

3. Data format of MRA

3.1 Whole structure

MRA is provided as multiple JSON files with the following folder structure.

```
mraData
 ├──── metaData.json
 ├──── definitions
 |     └──── definitions.json
 ├──── nodeProfile
 |     └──── 0x0EF0.json
 ├──── superClass
 |     └──── 0x0000.json
 └──── devices
       ├──── 0x0130.json
       ├──── 0x0290.json
       ├──── ...
```

"metaData.json" describes general information such as version and copyright.

"definitions.json" is a collection of the pairs of key (template name) and value (definition object) .
Each key-value pair describes a repeatedly used data expression.

Each file in the devices folder describes a device object. A file name represents EOJ of respective device such as 0x0130.json. In case a class name is changed in the APPENDIX new release, each device object is described in the separate file with a suffix to the file name such as 0x0288_01.json. The file with latest release version should have a file name without a suffix such as 0x0288.json.

"devices" is a collection of pairs of key (EOJ) and value (device description object). Each key-value pair describes a device object. <EOJ> is the first two bytes of ECHONET Object Code that identifies each ECHONET Lite object. For example, "0x0130" and "0x0290" represent an air conditioner and a lighting respectively. "0x0000.json" describes superclass of device object.

"0x0EF0.json" describes Node profile class and its superclass.

Files in devices folder such as 0x0EF0.json and 0x0000.json are described with a data format of a device description object.

3.2 meta data

Format

```
{
  "date": <release date>,
  "release": <ECHONET Lite release version>,
  "formatVersion": <data format version number>,
  "dataVersion": <MRA data version number>,
  "note": <any additional information>,
  "copyright": <copyright information>
}
```

Keyword	Type	Required	Description
date	string	yes	release date
release	string	yes	APPENDIX Release version (Example: M)
formatVersion	string	yes	Data format version (Example: 1.0.2)
dataVersion	string	yes	Data version (Example: 1.1.3)
note	object	no	Additional information
note.jp	string	no	Additional information (Japanese)
note.en	string	no	Additional information (English)
copyright	string	yes	copyright

Example

```
{
  "date": "2021-12-01",
  "release": "M",
  "formatVersion": "1.0.0",
  "dataVersion": "1.0.0",
  "note": {
    "ja": "Machine Readable APPENDIX 公式リリース",
    "en": "Machine Readable APPENDIX official release"
  },
  "Copyright": "(C) 2021 Kanagawa Institute of Technology, ECHONET CONSORTIUM"
}
```

3.3 device description object

Files in devices folder such as 0x0EF0.json and 0x0000.json are described with a data format of a device description object.

Format

```
{
  "eoj": <ECHONET object code>,
  "validRelease": {
    "from": <Release version>,
    "to": <Release version>
  },
  "className": {
    "ja": <class name in Japanese>,
    "en": <class name in English>
  },
  "shortName": <short name of the class name>,
  "elProperties": [
    <property description object>,
    <property description object>,
    ...
  ]
}
```

Keyword	Type	Required	Description	Example
eoj	string	yes	upper two bytes of EOJ in Hex expression	0x0130
validRelease	object	yes	range of valid release of APPENDIX	
validRelease.from	string	yes	beginning of the range	A, D
validRelease.to	string	yes	end of the range	L, latest (*1)
className	object	yes	class name defined in Appendix	
className.ja	string	yes	class name defined in Appendix in Japanese	家庭用エアコン
className.en	string	yes	class name defined in Appendix in English	Home air conditioner
shortName	string	yes	short name of the class name (*2)	homeAirConditioner
elProperties	array	yes	a collection of property description object (*3)	

*1 "latest" means the value of metaData.release
*2 "shortName" is a value utilized by the Device Description of the ECHONET Lite Web API
*3 When the properties described in Super class are overwritten by device object, it is described here

Example

```
{
  "eoj":"0x0130",
  "validRelease": {
    "from": "A",
    "to": "latest"
  },
  "className": {
    "ja": "家庭用エアコン",
    "en": "Home Air Conditioner"
  },
  "shortName": "homeAirConditioner",
  "elProperties": [
    <property description object for EPC:0xA0>,
    <property description object for EPC:0xA1>,
    ...
  ]
}
```

Multiple definitions for the same EOJ

In case a class name is changed in the APPENDIX release, each device object is described as separate file with a suffix to the file name such as 0x0288_01.json. The file with latest release version should have a file name without a suffix such as 0x0288.json.

Example

0x0288_1.json

```
{
  "eoj":"0x0288",
  "validRelease": {
    "from": "A",
    "to": "E"
  },
  "className": {
    "ja": "スマート電力量メータ", "en": "Smart Electric Energy Meter"
  },
  ...
}
```

0x0288.json

```
{
  "eoj":"0x0288",
  "validRelease": {
    "from": "F",
    "to": "latest"
  },
  "className": {
    "ja": "低圧スマート電力量メータ", "en": "Low-voltage Smart Electric Energy Meter"
  },
  ...
}
```

3.4 property description object

ECHONET property is described with a property description object.

Format

```
{
  "epc": <EPC>,
  "validRelease": {
    "from": <Release version>,
    "to": <Release version>
  },
  "propertyName": {
    "ja": <property name in Japanese>,
    "en": <property name in English>
  },
  "shortName": <short name of the property name>,
  "accessRule": {
    "get": <GET access rule>,
    "set": <SET access rule>,
    "inf": <Anno access rule>
  },
  "descriptions": {
    "ja": <description in Japanese>,
    "en": <description in English>
  },
  "atomic": <EPC of atomic operation>,
  "data": <EDT info in JSON schema>,
  "remark": {
    "ja": <remark in Japanese>,
    "en": <remark in English>
  },
  "note": {
    "ja": <note for DD in Japanese>,
    "en": <note for DD in English>
  }
}
```

Keyword	Type	Required	Description	Example
epc	string	yes	EPC in Hex expression	0x80
validRelease	object	yes	range of valid release of Appendix	
validRelease.from	string	yes	beginning of the range	A, D
validRelease.to	string	yes	end of the range	L, latest (*1)
propertyName	object	yes	property name defined in Appendix	
propertyName.ja	string	yes	property name in Japanese	動作状態
keyword	type	required	description	example

propertyName.en	string	yes	property name in English	Operation status
shortName (*2)	string	yes	short name of the property name	
accessRule (*3)	object	yes	access rule	
accessRule.get	string	yes	Get access rule	required
accessRule.set	string	yes	Set access rule	optional
accessRule.inf	string	yes	Anno access rule	required
descriptions	object	no	contents of property defined in Appendix	
descriptions.ja	string	no	description in Japanese	
descriptions.en	string	no	description in English	
atomic (*4)	string	no	EPC in HEX that requires atomic operation(SET) befor GET	
data	object	yes	data type of the property value described in chapter 4	
remark	object	no	remark defined in Appendix	
remark.ja	string	no	remark in Japanese	
remark.en	string	no	remark in English	
note (*2)	object	no	additional information for Device Description of ECHONET Lite Web API	
note.ja	string	no	note in Japanese	
note.en	string	no	note in English	

*1 "latest" means the value of metaData.release
*2 "shortName" and "note" are data utilized by the Device Description of the ECHONET Lite Web API
*3 A following table describes a relationship between APPENDIX (left and center columns) and accessRule of MRA.

Access rule	Required	accessRule
Get	blank	“get”:”optional”, “set”:”notApplicable”
Get	○	“get”:”required”, “set”:”notApplicable”
Set	blank	“get”:”notApplicable”, “set”:”optional”
Set	○	“get”:”notApplicable”, “set”:”required”

Access rule	Required	accessRule
Get/Set	blank	"get":"optional", "set":"optional"
Get/Set	○	"get":"required", "set":"required"
Get/Set	blank/○	"get":"optional", "set":"required"
Get/Set	○/blank	"get":"required", "set":"optional"

A following table describes a relationship between APPENDIX (left column) and accessRule of MRA.

Anno	accessRule
○	"inf":"required"
blank	"inf":"optional"

If there are some conditions about "required", use "required_c" instead of "required". In case a property comes under the conditionally required properties described in "Chapter 1 Outline of this document" of the APPENDIX, use "required_o" instead of "required".

*4 For example, in case of low-voltage smart electric energy meter (EOJ=0x0288), an operation of 'Set EPC=0xE5' is required before an operation of 'Get EPC=0xE2' (this kind of serial operation is called as atomic operation). In this case, "0xE5" is described in the atomic of EPC:0xE2.

Example


```
{
  "epc": "0x80",
  "validRelease": {
    "from": "A",
    "to": "latest"
  },
  "propertyName": {
    "ja": "動作状態",
    "en": "Operation status"
  },
  "shortName": "operationStatus",
  "accessRule": {
    "get": "required",
    "set": "required",
    "inf": "required"
  },
  "data": {
    "type": "state",
    "size": 1,
    "enum": [
      {
```

```
          "edt": "0x30",
          "name": "true",
          "descriptions": {
            "ja": "ON",
            "en": "ON"
          }
        },
        {
          "edt": "0x31",
          "name": "false",
          "descriptions": {
            "ja": "OFF",
            "en": "OFF"
          }
        }
      ]
    }
  }
```

Multiple definitions for the same EPC

In case property descriptions for the same EPC value are different between releases, enumerate them in an array at elProperties. A following example shows that the value of "data" is different between Release A-C and Release D-latest.

Example

```
  "elProperties": [
    {
      "epc":"0xE1",
      "validRelease": {
        "from": "A",
        "to": "C"
      },
      "propertyName": {
        "ja": "開度レベル設定", ...
      },
      "shortName": "...",
      "accessRule": {...
      },
      "data": {
        "type": "level",
        "base": "0x31",
        "maximum": 8
      }
    },
    {
      "epc":"0xE1",
```

```
      "validRelease": {
        "from": "D",
        "to": "latest"
      },
      "propertyName": {
        "ja": "開度レベル設定", ...
      },
      "shortName": "...",
      "accessRule": {...
      },
      "data": {
        "type": "number",
        "format": "uint8",
        "unit": "%",
        "minimum": 0,
        "maximum": 100
      }
    }
  ]
```

4. Data type of property value

ECHONET Data (EDT) is a binary data of property value. The meaning of each data depends on each property. Here are some examples of EDTs and possible meanings.

Example EDT=0x30

- Number: 48
- State: ON
- Level: 1

Example EDT=0xFFFF

- Number: 65535 (2byte unsigned)
- Number: -1 (2byte signed)
- Number: 255 and 255 (two 1byte data)
- Number: -1 and -1 (two 1byte data)
- State: Over Flow

In order to clarify the meaning of each property value, JSON schema like notation is utilized in the MRA. Data types are defined as follows.

Data Type	Description
number	number
state	describes a state or status
numericValue	coded number

Data Type	Description
level	value for a level
date	year, month and day
date-time	date and time
time	time
raw	data that can not be described by the other data types
bitmap	describes a state or number in bitmap
array	collection of data of the same data type
object	collection of data of key & value pair
oneOf	enumeration of data

- Data type "number", "state", "numericValue", "level", "date-time", "date" and "time" are called as primitive data type.
- Data type "bitmap", "array", "object" and "oneOf" are called as structured data type.
- Data type "raw" means that the data cannot be described by the other data types.
- Data type "bitmap" describes state(s) or number(s) in bitmaps.

4.1 number

Data type "number" describes numeric value. ECHONET Lite utilizes integer and fixed point number only. Data formats are as follows.

int8, int16, int32, uint8, uint16, uint32

ECHONET specifications (Part II Chapter 6.2.2) define specific values as codes which indicate "Underflow" or "Overflow" depending on data format.

Data format

data format	range	Underflow code	Overflow code
int8	-127 to 126	0x80	0x7F
int16	-32767 to 32766	0x8000	0x7FFF
int32	-2147483647 to 2147483646	0x80000000	0x7FFFFFFF
uint8	0 to 253	0xFE	0xFF
uint16	0 to 65533	0xFFFE	0xFFFF
uint32	0 to 4294967293	0xFFFFFFFE	0xFFFFFFFF

Format

```
{
  "type": "number",
  "format": <format of number>,
  "unit": <unit>,
  "minimum": <minimum number>,
  "maximum": <maximum number>,
  "enum": <array of possible values>,
  "multiple": <multiple value>,
  "multipleOf": <minimum digit>,
  "coefficient": [<EPC of coefficient>
  ],
  "overflowCode": <true or false>,
  "underflowCode": <true or false>,
  "descriptions": {
    "ja": <description in Japanese>,
    "en": <description in English>
  }
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"number"
format	string	yes	"int8", "int16", "int32", "uint8", "uint16", "uint32"
unit	string	no	unit
minimum	number	no	minimum number (*1)
maximum	number	no	maximum number (*1)
enum	array	no	restricting values by enumeration
multiple	number	no	multiple value (*2)
multipleOf	number	no	step (*3)
coefficient	array	no	EPCs for coefficient in Hex(string) (*4)
overflowCode	boolean	no	flag to utilize overflow code. default is true. (*5)
underflowCode	boolean	no	flag to utilize underflow code. default is true. (*5)
descriptions	object	no	descriptions
descriptions.ja	string	no	description in Japanese
descriptions.en	string	no	description in English

*1 The value of "minimum" and "maximum" are based on raw EDT value.
*2 “multiple” is also used to describe a number with fraction

> 12.5 = 125(EDT value) X 0.1(multiple)

*3 For example, if "multipleOf":2 is specified, it means 0,2,4, ... values. If "multiple" is specified, it means the step of the value before multiplication.

*4 For example, measured cumulative amount of electric energy (normal direction) (EPC:0xE0) of low-voltage smart electric energy meter (EOJ:0x0288) utilizes EPCs (0xD3 and 0xE1) as coefficients.

> "coefficient":["0xD3", "0xE1"]

*5 Though specific values are assigned to "Overflow" or "Underflow" code and a property utilizes limited range of values and never handle Overflow code or Underflow code, it can be described explicitly as follows.

> "overflowCode":false
> "underflowCode":false

Example

```
{
  "type": "number",
  "format": "int16",
  "unit": "A",
  "minimum": -9999,
  "maximum": 9999,
  "multiple": 0.1,
  "coefficient": [
    "0xA0",
    "0xA1"
  ]
}
```

4.2 state

Data type "state" describes a state or a status. For example, 0x30:ON, 0x31:OFF, 0x41:Heating.

Format

```
{
  "type": "state",
  "size": <data size>,
  "enum": [
    {
```

```
        "edt": <EDT value>,
        "name": <name of the state>,
        "descriptions": {
          "ja": <description in Japanese>,
          "en": <description in English>
        },
        "readOnly":true
      },
      ...
    ]
  }
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"state"
size	number	yes	data size. 0 in case of bitmap
enum	array	yes	array of state object

Schema definitions (state object)

Keyword	Type	Required	Description
edt	string	yes	EDT value in Hex
name (*1)	string	yes	a name of the state
descriptions	object	yes	descriptions of the state
descriptions.ja	string	yes	description in Japanese
descriptions.en	string	yes	description in English
readOnly (*2)	boolean	no	read only flag. default is false.

*1 "name" is not an item of APPENDIX, but it is defined in the MRA for the convenience.

*2 "readOnly" is set to "true" when it is utilized for the response of Get but not utilized for the response of Set. For example, Chamber temperature setting (0xE3) of electronic oven (0x03B8), "0x8002: Not specified". When access rules of property are GET only, "readOnly" should not be used.

Example

```
"data": {
  "type": "state",
  "size": 1,
```

```
    "enum": [
      {
        "edt": "0x41",
        "name": "true",
        "descriptions": {
          "ja": "沸き上げ中",
          "en": "Heating"
        }
      },
      {
        "edt": "0x42",
        "name": "false",
        "descriptions": {
          "ja": "非沸き上げ中",
          "en": "Not heating"
        }
      }
    ]
  }
```

4.3 numericValue

Data type “numericValue” assigns one value to another. For example, 0x00 means 1, 0x01 means 10 and 0x02 means 100.

Format

```
  {
    "type": "numericValue",
    "size": <data size>,
    "enum": [
      {
        "edt": <EDT value>,
        "numericValue": <numeric value>
      },
      ...
    ]
  }
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"numericValue"
size	number	yes	data size. 0 in case of bitmap
enum	array	yes	array of numericValue object

Schema definitions (numericValue object)

Keyword	Type	Required	Description
edt	string	yes	EDT value in Hex
numericValue	number	yes	numeric value

Example

```
{
  "type": "numericValue",
  "size": 1,
  "enum": [
    {
      "edt": "0x00",
      "numericValue": 1
    },
    {
      "edt": "0x01",
      "numericValue": 0.1
    },
    {
      "edt": "0x02",
      "numericValue": 0.01
    },
    {
      "edt": "0x03",
      "numericValue": 0.001
    },
    {
      "edt": "0x04",
      "numericValue": 0.0001
    }
  ]
}
```

4.4 level

Data type "level" describes a level value. For example, air flow level (EPC:0xA0) of a home air conditioner (EOJ:0x0130) defines 8 levels (1: week to 8: strong) and those are mapped to the value of 0x31 ... 0x38 respectively.

Format

```
{
  "type": "level",
```

```
    "base": <base value in Hex string>,
    "maximum": <maximum number of level>,
    "descriptions":{
      "ja": <description in Japanese>,
      "en": <description in English>
    }
  }
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"level"
base	string	yes	EDT value in Hex that is associated to LEVEL_1
maximum	number	yes	maximum level
descriptions	object	no	descriptions of the level (*1)
descriptions.ja	string	no	description in Japanese
descriptions.en	string	no	description in English

*1 This may be utilized to specify each item when multiple items of data type level are described with oneOf keyword.

Example: 8 levels by 0x31 to 0x38

```
{
  "type":"level",
  "base":"0x31",
  "maximum":8
}
```

4.5 date

Data type "date" describes date values (YYYYMMDD). When using some elements from the top such as year / month (YYYYMM) data, specify the data size with "size". If you do not specify the data size, the data will be 4 bytes.

Data Format

item	data size	minimum	maximum
Year	2	1	9999
Month	1	1	12

Day	1	1	31

Format

```
{
  "type":"date",
  "size": <data size>
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"date"
size	number	no	Data size in case of partial data

Example:

```
{
  "type":"date"
}
```

4.6 date-time

Data type "date-time" describes date and time values (YYYYMMDDHHMMSS). When using a part of the elements from the top, specify the data size with "size". If you do not specify the data size, the data will be 7 bytes of year, month, day, hour, minute, and second. If the data size is 4 bytes or less, use the data type "date".

Data Format

item	data size	minimum	maximum
Year	2	1	9999
Month	1	1	12
Day	1	1	31
Hour	1	0	23
Minute	1	0	59
Second	1	0	59

Format

```
{
  "type":"date-time",
  "size": <data size>
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"date-time"
size	number	no	Data size in case of partial data

Example: YYYY:MM:DD:hh:mm:ss

```
{
  "type":"date-time"
}
```

Example: YYYY:MM:DD:hh:mm (EOJ:0x0288, EPC:0xEC)

```
{
  "type":"date-time",
  "size":6
}
```

4.7 time

Data type "time" describes time values (HHMMSS). When using a part of the elements from the upper level such as hour and minute (HHMM) data, specify the data size with "size". If you do not specify the data size, it will be 3 bytes of hour, minute, and second data. Maximum value of Hour can be extended from a default value of 23 by using maximumOfHour.

Data Format

item	data size	minimum	maximum
Hour	1	0	23 (or 255)
Minute	1	0	59
Second	1	0	59

Format

```
{
  "type":"time",
  "size": <data size>,
  "maximumOfHour": < maximum value of Hour >
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"time"
size	number	no	Data size in case of partial data
maximumOfHour	number	no	Specifies maximum value of hour. default value is 23.

Example: HH:MM:SS

```
{
  "type":"time"
}
```

Example: HH(0 to 255):MM

```
{
  "type":"time",
  "size":2,
  "maximumOfHour":255
}
```

4.8 bitmap

Data type "bitmap" describes states and numeric values mapped to specific bitmap data.

Example: “bitmap” is 4-byte data. Utilizing the MSB of the first byte-data and lower 4 bits of the second byte. "index" specifies the byte number, and "bitmask" specifies the position of the bit in that byte.
Example:

```
    |<----------- 4 bytes ----------->|
    index:0  index:1  index:2  index:3
    XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX
    MSB  LSB MSB  LSB MSB  LSB MSB  LSB

    X******* ****XXXX ******** ********

    ^            ^^^^
    |            ||||
    |            ++++--- index = 1, bitmask = 0b00001111, data = 0...15
    +--- index = 0, bitmask = 0b10000000, data = 0 or 1
```

Format

```
{
  "type": "bitmap",
  "size": <size of total bitmaps data in bytes>,
  "bitmaps": [
    {
      "name": <bitmap name>,
      "descriptions": {
        "ja": <description of bitmap in Japanese>,
        "en": <description of bitmap in English>
      },
      "position": {
        "index": <index of the byte of a bitmap>,
        "bitMask": <bitmask to specify effective bits>
      },
      "value": <data type object>
    },
    ...
  ]
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"bitmap"
size	number	yes	size of total bitmaps data in bytes
bitmaps	array	yes	array of bitmap object

Schema definitions (bitmap object)

Keyword	Type	Required	Description
name	string	yes	bitmap name (*1)
descriptions	object	yes	descriptions of bitmap
descriptions.ja	string	yes	description in Japanese
descriptions.en	string	yes	description in English
Keyword	Type	Required	Description
position	object	yes	position of bitmap
position.index	number	yes	index of byte-data
position.bitMask	string	yes	bitmask to specify bits
value	object	yes	data type object of the bitmap value (*2)

*1 "name" is not an item of APPENDIX, but it is defined in the MRA for the convenience.
*2 In case of data type is "state", "size" in the "state" is 0.

Example: Water heater (EOJ:026B) Alarm status (EPC:0xC2)

> First byte, bit0: Out of hot water
> First byte, bit1: Water leaking
> First byte, bit2: Freezing

```
{
  "type": "bitmap",
  "size": 4,
  "bitmaps": [
    {
      "name": "noHotWater",
      "descriptions": {
        "ja": "湯切れ警報",
        "en": "Out of hot water"
      },
      "position": {
        "index": 0,
        "bitMask": "0b00000001"
      },
      "value": {
        "type": "state",
        "size": 0,
        "enum": [
          {
            "edt": "0x01",
            "name": "true",
            "descriptions": {
              "ja": "発生",
              "en": "Alarm"
            }
          },
          {
            "edt": "0x00",
            "name": "false",
            "descriptions": {
              "ja": "正常",
```

```
            "en": "No Alarm"
          }
        }
      ]
    }
  },
  {
    "name": "waterLeaking",
    "descriptions": {
      "ja": "漏水警報",
      "en": "Water leaking"
    },
    "position": {
      "index": 0,
      "bitMask": "0b00000010"
    },
    "value": {
      "type": "state",
      "size": 0,
      "enum": [
        {
          "edt": "0x01",
          "name": "true",
          "descriptions": {
            "ja": "発生",
            "en": "Alarm"
          }
        },
        {
          "edt": "0x00",
          "name": "false",
          "descriptions": {
            "ja": "正常",
            "en": "No Alarm"
          }
        }
      ]
    }
  },
  {
    "name": "waterFreezing",
    "descriptions": {
      "ja": "凍結警報",
      "en": "Water frozen"
    },
    "position": {
      "index": 0,
      "bitMask": "0b00000100"
    },
    "value": {
      "type": "state",
```

```
        "size": 0,
        "enum": [
          {
            "edt": "0x01",
            "name": "true",
            "descriptions": {
              "ja": "発生",
              "en": "Alarm"
            }
          },
          {
            "edt": "0x00",
            "name": "false",
            "descriptions": {
              "ja": "正常",
              "en": "No Alarm"
            }
          }
        ]
      }
    }
  ]
}
```

4.9 raw

Data type "raw" describes data that can not be described by other data types. This describes data size only. Please refer the Appendix (PDF) for details.

Format

```
{
  "type":"raw",
  "minSize": < minimum data size >,
  "maxSize": < maximum data size >
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"raw"
minSize	number	yes	minimum data size
maxSize	number	yes	maximum data size

Example: 8 bytes data

```
{
  "type":"raw",
  "minSize":8,
  "maxSize":8
}
```

Example: variable size data from 1 to 8 byte

```
{
  "type":"raw",
  "minSize":1,
  "maxSize":8
}
```

4.10 array

Data type "array" describes an array of items. Each item is the same data type. Data type of an item is primitive types (number, state, ...) or structured types (bitmap, array, object, oneOf). Data size of an array is calculated by multiplying size of an item and number of items.

Format

```
{
  "type":"array",
  "itemSize": < data size of each item>,
  "minItems": < minimum number of items>,
  "maxItems": < maximum number of items>,
  "items": <data type object of each item>
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"array"
itemSize	number	yes	data size of each item
minItems	number	no	minimum number of items
maxItems	number	yes	maximum number of items
items	object	yes	data type object

Example

```
{
  "type":"array",
  "itemSize":2,
  "maxItems":24,
  "items":{
    "type":"number",
    "format":"int16",
    "unit":"A",
    "minimum":-9999,
    "maximum":9999,
  }
}
```

4.11 object

Data type "object" describes structured data that consists of several elements. Data type of an element is primitive types (number, state, ...) or structured types (bitmap, array, object, oneOf). Data size of an object is sum of size of each element.

Format

```
{
  "type": "object",
  "properties": [
    {
      "elementName": {
        "ja": <element name in Japanese>,
        "en": <element name in English>
      },
      "shortName": <element name>,
      "element": <data type object>
    },
    ...
  ]
}
```

Schema definitions

Keyword	Type	Required	Description
type	string	yes	"object"
properties	array	yes	array of element

Keyword	Type	Required	Description
elementName	object	yes	element name
elementName.jp	string	yes	element name in Japanese
elementName.en	string	yes	element name in English
shortName (*1)	string	yes	short name of the element name
element	object	yes	data type object of the element

*1 "shortName" is utilized for Device Description of ECHONET Lite Web API.

Example: General Lighting, EPC=0xC0

```
{
  "type": "object",
  "properties": [
    {
      "elementName": {
        "ja": "R",
        "en": "R"
      },
      "shortName": "red",
      "element": {
        "type": "number",
        "format": "uint8",
        "minimum": 0,
        "maximum": 255
      }
    },
    {
      "elementName": {
        "ja": "G",
        "en": "G"
      },
      "shortName": "green",
      "element": {
        "type": "number",
        "format": "uint8",
        "minimum": 0,
        "maximum": 255
      }
    },
    {
      "elementName": {
        "ja": "B",
        "en": "B"
      },
      "shortName": "blue",
      "element": {
        "type": "number",
        "format": "uint8",
        "minimum": 0,
        "maximum": 255
      }
    }
  ]
}
```


Example: Low-voltage smart meter, EPC=0xE2

```
{
  "type": "object",
  "properties": [
    {
      "elementName": {
        "ja": "積算履歴収集日",
        "en": "day for which the historical data of measured cumulative amounts of
electric energy is to be retrieved"
      },
      "shortName": "day",
      "element": {
        "type": "number",
        "format": "uint8",
        "unit": "day",
        "minimum": 0,
        "maximum": 99
      }
    },
    {
      "elementName": {
        "ja": "積算電力量計測値",
        "en": "measured cumulative amounts of electric energy"
      },
      "shortName": "energy",
      "element": {
        "type": "array",
        "itemSize": 4,
        "minItems": 48,
        "maxItems": 48,
        "items": {
          "type": "number",
          "format": "uint32",
          "unit": "kWh"
        }
      }
    }
  ]
}
```

4.12 Mixed type: oneOf

In case data type definitions are different depending on the range of the value, enumerate them with a keyword "oneOf".

Format

```
{
  "oneOf":[
    <data type object>,
    <data type object>,
    ...
  ]
}
```

Example: 0x00 to 0x28: number, 0x31 to 0x38: level, 0x41: state

```
{
  "oneOf": [
    {
      "type": "number",
      "format": "uint8",
      "unit": "Celsius",
      "minimum": 0,
      "maximum": 40
    },
    {
      "type": "level",
      "base": "0x31",
      "maximum": 8
    },
    {
      "type": "state",
      "size": 1,
      "enum": [
        {
          "edt": "0x41",
          "name": "auto",
          "descriptions": {
            "ja": "自動",
            "en": "Automatic"
          }
        }
      ]
    }
  ]
}
```

5. definitions

For example, in the humidity measurement value (EPC: 0xBA) of a home air conditioner and the opening level setting (0xE1) of an electric blind, 1 byte of data expresses a value of 0-100%. In MRA, describe the following scheme in either case.

```
"data":{
  "type":"number",
  "format":"uint8",
  "unit":"%",
  "minimum":0,
  "maximum":100
}
```

This schema is, for example, used for Measured value of room relative humidity (EPC:0xBA) of home air conditioner class (EOJ:0x0130) or for Degree-of opening level (EPC:0xE1) of electrically operated blind/shade class (EOJ:0x0260). It is natural to define it once as a template and MRA refers this template when needed. Such template is descried in definition.json and MRA refers it by using a keyword “$ref”.

Format of definition.json

```
{
  "definitions":{
    <template name>:{ <schema> },
    ...
    }
}
```

Example of definitions.json

```
{
  "definitions": {
    "number_0": {
      "type": "number",
      "format": "uint8",
      "minimum": 0,
      "maximum": 0
    },
    "number_0-100percent": {
      "type": "number",
      "format": "uint8",
      "unit": "%",
      "minimum": 0,
      "maximum": 100
    },
    ...
  }
}
```

Example of 0x0130.json

```
{
  "eoj": "0x0130",
  "validRelease": {
    "from": "A",
    "to": "latest"
  },
  "className": {
    "ja": "家庭用エアコン",
    "en": "Home air conditioner"
  },
  "shortName": "homeAirConditioner",
  "elProperties": [
    {
      "epc": "0xBA",
      "validRelease": { ... },
      "propertyName": {
        "ja": "室内相対湿度計測値",
        "en": "Measured value of room relative humidity"
      },
      "accessRule": { ... },
      "descriptions": { ... },
      "data": {
        "$ref": "#/definitions/number_0-100%"
      }
    },

  ]
}
```

Note: The "$ref": "#/definitions/xxxx" path in the above example assumes the case where the data of the file provided as MRA is completed as JSON data having the following configuration.

```
{
  "metaData": { < content of metaData.json > },

  "definitions" : { < content of definitions.json > },
  "nodeProfile" : { < content of nodeProfile.json > },
  "superClass" : { < content of superClass > },
  "devices" : [
    { < content of 0x0130.json > },
    ...
    { < content of 0x0290.json > }
    ...
  ]
}
```

6. Node profile: 0x0EF0

"Node profile" describes both of super class of node profile and node profile n.

7. Super class

"Super class" describes superclass of device object.