Vol.2017-GN-100 No.32
Vol.2017-CDS-18 No.32
Vol.2017-DCC-15 No.32
2017/1/21

ECHONET 機器オブジェクト詳細規定のデータ化

藤田裕之[†1]　杉村博[†1]　村上隆史[†1]　一色正男[†1]

概要：ECHONET Lite 対応機器の各機器の機能を記述した「ECHONET 機器オブジェクト詳細規定」の内容を整理し JSON フォーマットでデータ構造やデータの値を定義した．その定義に則り「ECHONET 機器オブジェクト詳細規定」Release G に記述されている全ての情報（178 機種，921 機能）をデータ化した．さらにそのデータ内容の確認と評価のために iOS 用のビューワーアプリを開発した．このデータファイルを利用することで ECHONET Lite を利用したコントローラやサービスの開発が効率的になる．

キーワード：エコーネットライト，ECHONET Lite，HEMS， JSON， iPhone，iOS，ツール

Creation of Data File from the documentation of “Detailed Requirements for ECHONET Device objects”

HIROYUKI FUJITA[†1]　HIROSHI SUGIMURA[†1]
TAKASHI MURAKAMI[†1]　MASAO ISSHIKI[†1]

Abstract: We have reviewed contents of the document "Detailed Requirements for ECHONET Device objects" that describes details of every function of every ECHONET device, and defined data structure and data values in JSON format. Based on the definition, we have created a data file that describes all information in the document of Release G that includes 178 devices and 921 functions. And also we developed viewer software for iOS to verify and evaluate the data. We are sure this data file can accelerate the development of controllers and services utilizing ECHONET Lite.

Keywords: ECHONET Lite, HEMS

1. はじめに

HEMS の基盤となる ECHONET Lite プロトコルの仕様は通信パケットの仕様と各機器の機能詳細の仕様で構成されている．前者は ECHONET Lite SPECIFICATION（第 1 部～第 5 部）[1]に記述されており、後者は「ECHONET SPECIFICATION APPENDIX ECHONET 機器オブジェクト詳細規定」[2]に記述されている．ECHONET Lite を利用してコントローラやサービスを開発する場合は，この「ECHONET 機器オブジェクト詳細規定」を参照して送信データのパラメータ設定や,受信データ解析等を実施する．

例えばエアコンの運転モードを冷房に変更したい場合は，まず機器オブジェクト一覧から「家庭用エアコン」を検索し，該当するページを参照し，対応する機器オブジェクトコード（0x0130）を取得する．次に家庭用エアコンのプロパティから運転モードを検索し対応するプロパティコード（0xB0）を取得する．運転モードのプロパティ内容から冷房に対応するコード（0x42）を取得する．これらのデータから EOJ=0x0130, EPC=0xB0, EDT=0x42 として ECHONET Lite パケットを作成し送信する．

ECHONET 機器オブジェクト詳細規定は PDF のみで配布されている．記述されている膨大な量の情報(a)はバイナリーデータに関する記述である．プログラムから容易に利用できるような API や SDK はエコーネットコンソーシアムからは提供されていない．ECHONET 機器オブジェクト詳細規定に記述されている情報はプログラムから直接は利用できないため，コントローラやサービスのプログラム開発時には「ECHONET 機器オブジェクト詳細規定」を目で見てデータを手入力する必要があるという非効率な開発環境である．

従来,「ECHONET 機器オブジェクト詳細規定」をデータ化する試みは Sony Computer Laboratory などで行われてきた(3)が，後述するように機能毎の状態表現や制御パラメータのデータ表現があまりにも多彩であるために，データ化は機器オブジェクトコードやプロパティコードなど表現しやすい一部の情報に限られていた．また ECHONET Lite 用 SDK を市販しているベンダーは SDK の内部にデータベースを構築しているが，それらのデータはオープンになっていない．

ECHONET Lite を利用したコントローラやサービスの開発を促進するためには，ECHONET 機器オブジェクト詳細規定に記述されている情報をオープンな形で利用できるようにする必要がある．本研究では特に機能毎のプロパティ

†1 神奈川工科大学
Kanagawa Institute of Technology

a) Release G の場合，総ページ数は 493 ページ，178 機種，921 機能に関して記述されている

Vol.2017-GN-100 No.32
Vol.2017-CDS-18 No.32
Vol.2017-DCC-15 No.32
2017/1/21

内容に記述されている状態表現や制御パラメータのデータ構造とデータ表現を検討した．この検討結果を基にして「ECHONET 機器オブジェクト詳細規定 Release G」の内容を全てデータ化し，GitHub で公開した(b)．さらにこのデータの有用性を確認するために，iOS 用のビューワーアプリを開発し，ソースコードを GitHub で公開した(c)．

2. 解決する課題

ECHONET 機器オブジェクト詳細規定の内容を家庭用エアコンに関する記述（図１）を例にして説明する．クラスグループコード（0x01）とクラスコード（0x30）を組み合わせた２バイトデータ（0x0130）で家庭用エアコンを表現し，これを ECHONET Object（EOJ）と呼ぶ．プロパティ名称欄にはデバイスが持っている機能の名前が記述されている．EPC 欄にはプロパティ毎に割り振られた 1 byte の値が記述されている．EPC は ECHONET Property Code の略称である．プロパティ内容の欄にはプロパティの定義と取りうる値が簡単に記述されている．詳細な内容はテーブルの後に記述されている．データ型の欄には C 言語の型表現を利用してデータの型が記述されている．データサイズの欄にはデータのサイズがバイト単位で記述されている．単位の欄にはデータの単位と倍率が記述されている．アクセスルールの欄には，そのプロパティに対してアクセスできる命令の種類が記述されている．必須の欄にはそのプロパティの実装が必須であるか否かが記述されている．状変時アナウンスの欄には，プロパティ値の変化時にデバイスから自発的に「状変時アナウンス」を送信する機能の実装が必要であるか否かが記述されている．備考欄には主にアクセスルールに関する条件が記述されている．

３．２．１ 家庭用エアコンクラス規定

クラスグループコード　：0x01
クラスコード　：0x30
インスタンスコード　：0x01～0x7F (0x00：全インスタンス指定コード)

プロパティ名称	EPC	プロパティ内容 値域(10 進表記)	データ型	データサイズ	単位	アクセスルール	必須	状変時アナウンス	備考
動作状態	0x80	ON／OFF の状態を示す。 ON＝0x30，OFF＝0x31	unsigned char	1 Byte	－	Set Get	○ ○	○	
節電動作設定	0x8F	機器の節電動作を設定し、状態を取得する。 節電動作中=0x41 通常動作中=0x42	unsigned char	1 Byte	－	Set/ Get	○	○	
運転モード設定	0xB0	自動／冷房／暖房／除湿／送風／その他の運転モードを設定し、設定状態を取得する。 順番に以下のコードが対応。 0x41/0x42/0x43/0x44/0x45/0x40	unsigned char	1 Byte	－	Set/ Get	○	○	
温度自動設定	0xB1	AUTO／非 AUTO を設定し、設定状態を取得する。 AUTO＝0x41,非 AUTO＝0x42	unsigned char	1 Byte	－	Set/ Get			
急速動作モード設定	0xB2	通常運転／急速／静音を設定し、設定状態を取得する。 通常運転＝0x41 急速＝0x42，静音＝0x43	unsigned char	1 Byte	－	Set/ Get			
温度設定値	0xB3	温度設定値を設定し、設定状態を取得する。	unsigned char	1 Byte	℃	Set/ Get	○		

図 1　ECHONET 機器オブジェクト詳細規定の例

これらの情報を，プログラムで利用しやすい形にデータ化することが本課題である．以下に課題を掘り下げて説明する．

記述されている内容を整理すると大きく分けて基本項目，アクセスルールに関する項目，プロパティ内容に関する項目に分類できる．

- 基本項目：プロパティ名称，EPC
- アクセスルールに関する項目：アクセスルール，必須，状変時アナウンス，備考
- プロパティ内容に関する項目：プロパティ内容，データ型，データサイズ，単位

基本項目に関しては記述されたデータをそのまま利用すればいい．

アクセスルールに関しては，デバイスが対応するプロパティを実装している場合に ECHONET Lite の３種類の命令（Set, Get, 状変時アナウンス）のどれに対応するか，またその命令の対応が必須であるかどうかに関して情報が整理されておらず統一的な表現となっていないことが課題である．

プロパティ内容に関する項目に関しては、更に具体例を挙げて説明する．

0x0130：家庭用エアコン

#	*EPC　プロパティ名称*	*プロパティ内容*
1	0x80：動作状態	0x30：ON, 0x31：OFF
2	0xB4：相対湿度設定値	0～100%
3	0xBF：相対温度設定値	（-127～125）x 0.1℃
4	0xB3：温度設定値	0～50℃, 0xFD:設定値不明
5	0xC6：搭載空気清浄方法	bit0: 電気集塵方式 （0: 非搭載, 1: 搭載） bit1: クラスタイオン方式 （0: 非搭載, 1: 搭載）
6	0x91：タイマ時刻設定値	1byte 目時 2byte 目分

0x0EF0：ノードプロファイル

#	*EPC　プロパティ名称*	*プロパティ内容*
7	0x83：識別番号	17byte の任意の値

0x0288：低圧スマート電力量メータ

#	*EPC　プロパティ名称*	*プロパティ内容*
8	0xEA：積算電力量計測値	YYMDHMS(7byte) + 電力量(4byte)
9	0xE2：積算電力量履歴	DAY(2byte) + 電力量(4byte) x 48 回

具体例の解説

#1　ON/OFF の状態を 0x30 と 0x31 の値に割り当てる Key & Value データ．

#2　数値データ．値域(0～100)と単位（%）をもつ．

#3　正負の値をとる数値データ．値域（-127～125）と単位

b https://github.com/KAIT-HEMS/ECHONET-APPENDIX
c https://github.com/KAIT-HEMS/ELViewer

Vol.2017-GN-100 No.32
Vol.2017-CDS-18 No.32
Vol.2017-DCC-15 No.32
2017/1/21

（℃）と倍率（x0.1）をもつ.
#4　数値データと Key & Value データの両方を取りうる.
#5　bitmap 形式のデータ
#6　時刻データ
#7　任意のバイナリーデータ
#8　複数の種類（年月日時分秒と電力量）のデータ
#9　一部のデータ（電力量）の繰り返し

以上のようにプロパティの内容は様々な種類のデータが存在する．また一つのプロパティの中で複数の種類のデータが存在する．このようなさまざまなデータを記述できるデータ表現方法とデータ構造を定義することが課題である.

3. 課題解決の方針

ECHONET Lite を利用したコントローラやサービスのプログラム開発時に容易に利用できることを前提に「ECHONET 機器オブジェクト詳細規定」に記述されているできるだけ多くの情報をデータ化することを基本方針とする.「ECHONET 機器オブジェクト詳細規定」に記載されている全ての機器オブジェクトの全ての機能（Release G には 178 の機器オブジェクト，921 の機能について記述されている）をレビューした上でデータ化の詳細を決定する.

データ化の形式としては，さまざまなプログラム言語から容易に利用でき，Web アプリとの親和性が高く，またデータ表現の柔軟性を考慮して JSON フォーマットを採用することにした.

図２に示すように機器オブジェクトは全ての機器で共通なスーパークラスと機器毎に異なる機器オブジェクト詳細規定からなる．またノードプロファイルのスーパークラスは機器オブジェクトのスーパークラスと同一ではない．そこで今回は以下の３つの JSON データファイルを作成する.

- ノードプロファイル用 JSON file：
  ノードプロファイルのスーパークラスと機器オブジェクト情報を記述したもの
- スーパークラス用 JSON file：
  ノードプロファイル以外の機器オブジェクトのスーパークラスの情報を記述したもの
- 機器オブジェクト用 JSON file：
  ノードプロファイル以外の機器オブジェクトの情報を記述したもの

プログラムでのパースを容易にするために全く同じ書式，構造でデータファイルを作成する.

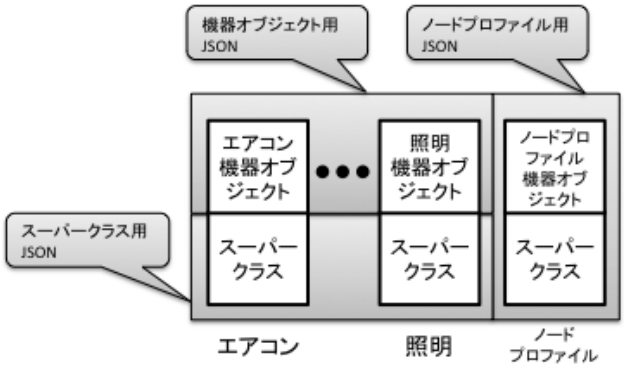

図 2　機器オブジェクトとスーパークラス

4. 提案するデータ化について

まずデータファイルの全体構成を説明する．それからアクセスルールとプロパティデータに関して説明する.

4.1 全体の構成

JSON データの全体構成を以下に示す.

```
{
    "version":"G",
    "date":"2016/8/23",
    "elObjects":{ ... }
}
```

“version”は「ECHONET 機器オブジェクト詳細規定」の Release version を示す．”date”はデータ作成日を示す．”elobjects”は以下のように EOJ のコードを KEY として機器オブジェクトを列挙する.

```
"elObjects":{
    "0x0130":{ ... }    // 家庭用エアコン
    "0x0260":{ ... }    // 電動ブラインド
    "0x0288":{ ... }    // 低圧スマートメータ
    "0x0290":{ ... }    // 一般照明
    "0x03B7":{ ... }    // 冷凍冷蔵庫
    ...
}
```

機器オブジェクトはオブジェクト名と EPC の列挙で構成される.

```
"0x0130":{
    "objectName":"家庭用エアコン",
    "epcs":{
        "0xB0":{ ... }   // 運転モード設定
        "0xB1":{ ... }   // 温度自動設定
        "0xB2":{ ... }   // 急速動作モード設定
        "0xB3":{ ... }   // 温度設定値
        ...
        }
    }
```

EPC は基本項目とアクセスルールに関する項目が記述された後にプロパティ内容に関する項目が”edt”として記述される.”epcName”は EPC の名前を示す.”epcSize”はこの EPC に対応する EDT のデータサイズを示す．”accessMode”及び”edt”に関しては次節以降で解説する.

```
"0xB3":{
```

Vol.2017-GN-100 No.32
Vol.2017-CDS-18 No.32
Vol.2017-DCC-15 No.32
2017/1/21

```
"epcName":"温度設定値",
"epcSize":1,
"accessModeSet":"required",
"accessModeGet":"required",
"accessModeAnno":"notApplicable",
"accessModeCondition":"-",
"edt":[<ELEMENT1>, <ELEMENT2 >, ... ]
}
```

4.2 アクセスルールに関する項目

アクセスルールに関しては，Set, Get, 状変時アナウンスそれぞれに対して実装の必要性を以下のクライテリアで記述する．

- 必須：required
- 必須ではない：optional
- 適応外：notApplicable

備考欄の情報は accessModeCondition として string で表現する．以下にサンプルを示す．

```
{
  "accessModeSet":" optional",
  "accessModeGet":"required",
  "accessModeAnno":" notApplicable",
  "accessModeCondition":"-"
}
```

4.3 プロパティ内容に関する項目

プロパティ内容に関しては，様々な種類のデータを表現する方法と，繰り返しも含めて複数のデータを表現するデータ構造について述べる

4.3.1 データ表現

「ECHONET 機器オブジェクト詳細規定 Release G」に記述されているプロパティの内容を全てレビューしデータ表現を以下のように整理した．

a) Key & Value 値:

解説：個々の状態に数値を割り当てる場合．

（例）ON=0x30, OFF=0x31

表現方法：JSON の object 形式で Key と Value を列挙する．

（例）”keyValues”:{"0x30":"ON", "0x31":"OFF"}

b) 数値データ:

解説：数値そのものに意味がある場合．

（例）25%, -5℃

表現方法：数値データは以下の項目で記述する．

- "integerType": 数値が符号なし"Unsigned"か，符号付き"Signed"かを示す
- "magnification": 数値データに対する倍率を 10 の N 乗表記した指数部. 省略可. 例: 100 倍の場合: 2, 0.1 倍の場合:-1
- "unit": 単位．省略可．例: "℃"
- "min": 値域の最小値，例: 0
- "max": 値域の最大値，例: 100

（例）”numericValue”:{ "integerType":"Unsigned", "magnification":-1, "unit":"A", "min":0, "max":1000}

Unsigned data．値域は 0A から 100.0A．倍率は 0.1 倍．

c) レベルデータ:

解説:制御のレベルをある範囲の値に対応させる場合．

（例）レベル 1～レベル 8 を 0x31～0x38 に対応させる

表現方法：最小値(min)と最大値(max) を object 形式でで記述する．

（例）” level “:{"min":49, "max":56}

d) ビットマップデータ:

解説：bit 毎に動作を割り当てる場合．例：搭載空気清浄方法 bit0（電気集塵方式）：非搭載=0, 搭載=1, bit1（クラスタイオン方式）：非搭載=0, 搭載=1

表現方法：bitmap は以下の項目で記述する．

"bitName"：名前．例: "電気集塵方式"

"bitRange"：対応する bit の番号．複数 bit を指定する場合は列挙する．例: [0], [0,1,2]

"bitValues"：bit(s)の値と Value を object 形式で記述する．例: {"0":"ON", "1":"OFF"}

（例）” bitmap”:{“bitName”:” 制御レベル”, “bitRange”:[0,1,2], “bitValues”:{“0”:”level0”, “1”:”level1”, “2”:”level2”, “3”:”level3”, “4”:”level4”, “5”:”level5”, “6”:”level6”, “7”:”level7”}}

e) RAW データ:

解説：データを意味のある数値としてではなく，値そのものを利用する場合．例: 製造番号

表現方法:rawData の type を記述する．"binary", "ASCII", "ShiftJIS"を定義している．

（例）"rawData":"binary"

f) カスタムタイプ:

解説：複数のデータの組み合わせで特定の意味を持たせる場合. 例: 年月日（2byte, 1byte, 1byte），日時（1byte, 1byte）

表現方法：custom type として以下のものを定義している．

YYM: Year(2byte), Month(1byte)

YYMD: Year(2byte), Month(1byte), Day(1byte)

HM: Hour(1byte), Minute(1byte)

HMS: Hour(1byte), Minute(1byte), Second(1byte)

HMF: Hour(1byte), Minute(1byte), Frame(1byte)

MS: Minute(1byte), Second(1byte)

（例）” customType”:” YYMD”

g) その他のデータ:

解説:特定の EPC 固有のデータ表現の場合はそれを抽象化して表現してもプログラムとしては再利用できないので意味がない．また単にプレースホルダーとしての「予約済み」も含む．

例：DR イベントコントローラの EPC=0xE0（イベント

情報）
表現方法：others として以下のものを定義している．
"referSpec": 仕様書参照
"reserved": 予約済
（例）” others”:"referSpec"

4.3.2 プロパティ内容のデータ構造

前節ではさまざまな種類のデータの表現方法を定義した．本節では，一つのプロパティの中で複数種類のデータを扱う場合の記述方法を定義する．

a) 一つのデータに複数のデータ表現が混在する場合

例えば家庭用エアコン（EOJ:0x0130）の室内温度計測値（EPC: 0xBB）のプロパティ内容は以下のように定義されている．

-127℃～125℃
overflow: 0x7F, underflow: 0x80, 計測不能:0x7E

これは室内温度計測値が値によって「数値データ」の場合と「Key&Value 値」の場合があることを示している。このように一つのデータに複数のデータ表現が混在する場合は，以下のように列挙して記述する．

```
"content":{
    ”numericValue”:{ ... },
    ”keyValues”:{ ... }
    }
```

b) 一つのプロパティの中に意味・タイプ・サイズなどが異なるデータが複数存在する場合

例えば低圧スマート電力量メータ（EOJ:0x0288）の定時積算電力量計測値（EPC:0xEA）のプロパティ内容は以下のように定義されている．

１～７バイト目：年月日時分秒（YYMDHMS）
８～１１バイト目：積算電力量計測値（0～99,999,999）

このように一つのプロパティの中に意味・タイプ・サイズなどが異なるデータが複数存在する場合もある．そこで意味を持つ最小のデータの単位として ELEMENT という概念を導入する．上記の例は，定時積算電力量計測値というプロパティが年月日時分秒という ELEMENT１と積算電力量計測値という ELEMENT２で構成されているということである．

プロパティデータ（edt）は，ELEMENT の配列として記述する．ELEMENT の記述方法は次節で説明する．

```
"edt":[
    { ELEMENT 1 の記述 },
    { ELEMENT 2 の記述 },
    { ELEMENT 3 の記述 },
   ...
    ]
```

c) ELEMENT の繰り返しがある場合

例えば低圧スマート電力量メータ（EOJ:0x0288）の積算電力量計測値履歴１（EPC:0xE2）のプロパティ内容は以下のように定義されている．

１～２バイト目：積算履歴収集日（0～99）
３バイト目以降：積算電力量計測値（0～99,999,999）の 48 回繰り返し

このように，ある ELEMENT が繰り返される場合もある．そこで各 ELEMENT の記述を ELEMENT 名，ELEMENT サイズ, ELEMENT の繰り返し回数，コンテンツで定義する．上記の例は，このように記述できる．

第１ELEMENT

```
{
    "elementName":"積算履歴収集日",
    "elementSize":1,
    "repeatCount":1,
    "content":{
        "numericValue":{
            "integerType":"Unsigned",
            "magnification":0,
            "unit":"day",
            "min":0, "max":99
        }
    }
}
```

第２ELEMENT

```
{
    "elementName":"積算電力量計測値",
    "elementSize":4,
    "repeatCount":48,
    "content":{
        "numericValue":{
            "integerType":"Unsigned",
            "magnification":0,
            "unit":"kWh",
            "min":0, "max":99999999
        }
    }
}
```

以上の定義をもとに、図３に JSON データの例を示す.

Vol.2017-GN-100 No.32
Vol.2017-CDS-18 No.32
Vol.2017-DCC-15 No.32
2017/1/21

```jason
{
    "version":"G",
    "date":"2016/8/23",
    "elObjects":{
        "0x0130":{
            "objectName":"家庭用エアコン",
            "epcs":{
                "0xB0":{ ... }
                ...
                "0xB3":{
                    "epcName":"温度設定値",
                    "epcSize":1,
                    "accessModeSet":"required",
                    "accessModeGet":"required",
                    "accessModeAnno":"required",
                    "accessModeCondition":"-",
                    "edt":[
                        {
                            "elementName":"温度設定値",
                            "elementSize":1,
                            "repeatCount":1,
                            "content":{
                                "keyValues":{"0xFD":"設定値不明"}
                                "numericValue":{
                                    "integerType":"Unsigned",
                                    "magnification":0,
                                    "unit":"℃",
                                    "min":0, "max":50
                                    }
                                }
                            }
                        }
                    ]
                }
                ...
            }
        }
    }
}
```

図 3　JSON データ

5. Data Viewer アプリ

作成した JSON ファイルを利用して iPhone 用の Data Viewer アプリを開発した．JSON ファイルの構造が使い易い形式になっているかを確認するためと，データそのものをチェックするためである．また UDP で受信したデータを解析する場合などにリファレンスブック代わりとしても利用できるという意味で実用的なアプリでもある．

ソースコードは GitHub で公開している．画面表示は全て TableView を利用し，NavigationViewController で画面遷移を行なっている．

図 4 ～図 6 にアプリケーションのキャプチャ画面を示す．

図 4 は TOP 画面である．ここで機器オブジェクト，Node Profile，Super Class を選択する．この 3 種類のデータは別々の JSON file であるが，データ構造を全く同一にしたおかげで，プログラム中ではファイル名以外まったく同じコードで記述できた．

図 5 は機器オブジェクトの選択画面である．178 機種の候補から選択する．

図 6 は選択された機器オブジェクトの EPC 選択画面である．表示される EPC は選択した機器オブジェクトに対応したものである． EPC の総数はのべで 921 種類である．

図 7 は選択された EPC の詳細データである．最初のセクションに名前やアクセスモードなどの共通項目が記述されている．それ以降のセクションには各エレメント毎の情報が記述されている．

図 4　TOP 画面　　図 5　機器オブジェクト選択画面

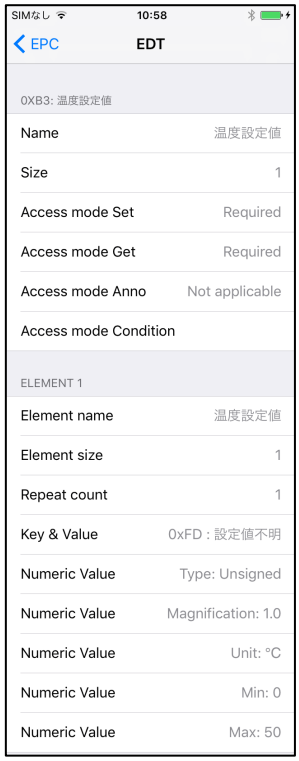

図 6　EPC 選択画面　　図 7　EDT 選択画面

6. 結論

ECHONET 機器オブジェクト詳細規定に記述されている膨大な情報をもとに，JSON フォーマットを利用してプログラムで利用しやすいデータファイルを作成することができた．これにより ECHONET Lite を利用したアプリケーションやサービス開発の促進が期待される．

Vol.2017-GN-100 No.32
Vol.2017-CDS-18 No.32
Vol.2017-DCC-15 No.32
2017/1/21

参考文献

1. “ECHONET Lite SPECIFICATION Version 1.12”
http://echonet.jp/spec_v112_lite/
2 “ECHONET 機器オブジェクト詳細規定 Release G Revised”
http://echonet.jp/spec_object_rg_revised/
3 ECHONETLite-ObjectDatabase,
https://github.com/SonyCSL/ECHONETLite-ObjectDatabase