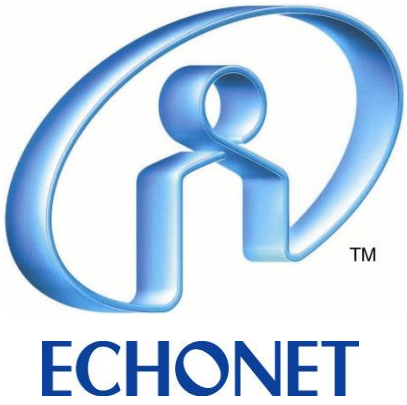
ECHONET Lite Web API
ガイドライン

Version 1.00



© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

改定履歴

日付	版	説明
2018/10/03	Ver.1.00	制定、一般公開

- エコーネットコンソーシアムが発行している規格類は、工業所有権(特許,実用新案など)に関する抵触の有無に関係なく制定されています。
  エコーネットコンソーシアムは、この規格類の内容に関する工業所有権に対して、一切の責任を負いません。
- この書面の使用による、いかなる損害も責任を負うものではありません。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

目次

第１章 はじめに 1-1

１．１ 用語 1-2
１．２ 参照規格 1-2

第２章 ECHONET Lite Web API の対象範囲 2-1

第３章 ECHONET Lite Web API のユースケース 3-1

３．１ 状態取得・制御・通知 3-1
３．２ 機器の一覧取得・管理情報取得 3-1
３．３ 機器の一括操作 3-2
３．４ 仮想機器 3-2
３．５ EMS 情報、サーバ蓄積ログ 3-3
３．６ 認証・認可 3-4
３．７ 対象ユースケースの絞り込み 3-4

第４章 Web API モデルの指針 4-1

４．１ 基本方針 4-1
４．２ アプリケーション名 4-2
４．３ API バージョン 4-2
４．４ バージョン一覧取得 4-3
４．５ 対象リソース種一覧取得 4-4
４．６ 機器一覧取得 4-7
４．７ 機器情報（Device Description）取得 4-9
４．８ 機器オブジェクトのプロパティ値操作（SET/GET など） 4-11
４．９ クライアントのキャッシュの扱い 4-12
４．１０ 機器オブジェクトのプロパティ値通知（INF） 4-13
４．１１ 認証・認可 4-18

第５章 ECHONET Lite 仕様のマッピング指針 5-1

５．１ ECHONET Lite フレームのマッピング 5-1
５．２ DEOJ のマッピング 5-1
５．３ ESV のマッピング 5-2
５．４ EPC、EDT のマッピング 5-3
５．５ ECHONET Lite 機器オブジェクトのマッピング方式および操作 5-3
５．６ エラー処理 5-10
５．７ 機器情報 5-13

第６章 おわりに 6-1

第７章 謝辞 7-1

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

図目次

図 1-1　本ガイドラインの想定モデル 1-1
図 2-1　本ガイドライン対象のシステム構成図 2-1
図 3-1　ユースケース：状態取得・制御・通知 3-1
図 3-2　ユースケース：機器の一覧取得・管理情報取得 3-2
図 3-3　ユースケース：一括操作 3-2
図 3-4　ユースケース：仮想機器 3-3
図 3-5　ユースケース：EMS 情報、サーバ蓄積ログ 3-4
図 3-6　ユースケース：認証・認可 3-4
図 4-1　サーバが提供するリソース種の考え方 4-6
図 4-2　MQTT による INF 実現例 4-17

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

表目次

表 3-1　各ユースケースの本ガイドライン対応方針 3-5
表 4-1　機器一覧取得時の応答内容 4-8
表 4-2　機器情報取得時の応答内容 4-10
表 4-3　Property Object の内容 4-11
表 5-1　ECHONET Lite フレームの対応 5-1
表 5-2　ESV の対応 5-2
表 5-3　コード指定によるリクエスト内容 5-9
表 5-4　コード指定によるレスポンス内容 5-9
表 5-5　HTTP Status Code 5-11
表 5-6　HTTP Status Code の活用事例 5-11
表 5-7　サービスレベルのエラー応答 5-12
表 5-8　エラータイプの種類 5-12

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

第１章 はじめに

本書は、ECHONET Lite 規格を活用し Web API によるサービス提供を実現するために考慮すべき観点や参考事例についてまとめたガイドラインとなる。ECHONET Lite のモデルを、サーバ環境を介してクライアント等へ提供し、提携サービスや応用アプリケーションの開発など利用促進に繋げるべく、Web API を用いたシステム構築やアプリケーション実装の際に参考にすべき指針について整理する。本書の提供により、エコーネットの更なる普及・拡大が進むことを期待する。

以降、第２章で本ガイドラインが対象とする構成について定め、第３章で検討候補となるユースケースについて整理する。第４章では、Web API 化における基本方針として、リソース設計やインタフェース設計、認証・許可方式などについて、第５章では、ECHONET Lite 仕様をベースに Web API への対応（変換）ルールやエラー表現などについて述べる。

本書は設計指針を示すガイドラインの位置づけとし、Web API 化される具体的な機器オブジェクトのデータ型や（プロパティに相当する）各種サービス定義などは、Appendix として別冊にて提供する。

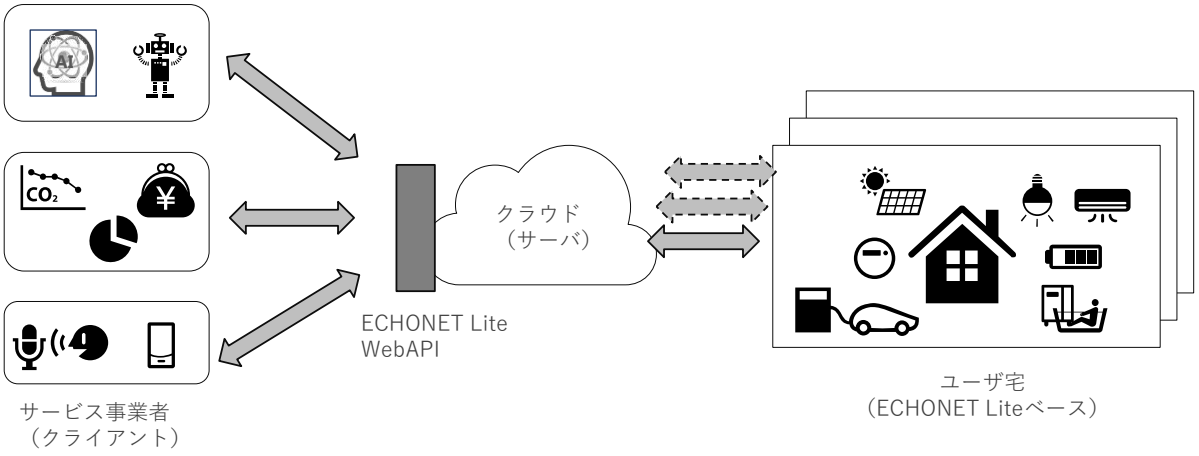


図 1-1　本ガイドラインの想定モデル

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

１．１ 用語

Web API	HTTP(S)プロトコルを用いたシステム間のインタフェースであり、リクエスト／レスポンス方式でデータをやりとりする。本書では、特に呼び出される側のシステム（サーバ側）で提供されるものを指す。
RESTful	URI と HTTP メソッドを用いて操作するリソースを特定する設計原則
JSON	軽量なテキストベースの言語非依存データ交換形式
OAuth2.0	権限の認可(authorization)を行うためのオープンスタンダード

１．２ 参照規格

本仕様で参照する規格を以下に挙げる。本仕様書に明示的な説明がない事柄については、規格文書に従う。

[EL] The ECHONET Lite Specification Version 1.01 以降
[ELOBJ] ECHONET Specification APPENDIX: ECHONET 機器オブジェクト詳細規定 Release J 以降
[HTTP] Hypertext Transfer Protocol (HTTP/1.1) RFC 7230～7235, https://tools.ietf.org/html/rfc7230 ～ 7235
[JSON] The JavaScript Object Notation (JSON) Data Interchange Format RFC 8259, https://tools.ietf.org/html/rfc8259
[OAUTH] OAuth 2.0 for Native Apps (RFC 8252), The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC6750)

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

第２章 ECHONET Lite Web API の対象範囲

本章では、ECHONET Lite Web API ガイドライン第１版の策定にあたり、対象範囲として想定するシステム構成について図 2-1 に示す。

基本的には、1 つの宅内に（1 台以上複数可の）ECHONET Lite 対応コントローラが存在し、その配下に ECHONET Lite 対応の機器が複数台接続されるモデルを想定する。コントローラと機器の組み合わせは複数組存在しても良い。通常はコントローラを介してベンダ保有のサーバ X（クラウド）へ接続されるケースが想定されるが、コントローラを介さず直接サーバ X へ接続可能な場合も対象に含んで良い。さらに、非 ECHONET Lite 機器であってもサーバ X へマッピングすることで、本書で示す Web API 形式のモデルへ変換可能であれば、こうした機器も対象可能とする。これらコントローラや機器をどのようにサーバ X 上へマッピングするかについては、実装依存であるとし、本ガイドラインのスコープ外とする。

また、本書では、サーバ X は事前にサービス・アプリ提供者であるクライアント A を知っており、適切な認証・認可を与えている（もしくは与えることが可能である）ことを前提としている。具体的な認証・認可手順は、サーバ毎の実装形態にて規定する事項とし、事例紹介に留める。

クライアント A、クライアント B などのクライアント間でのネゴシエーションや、サーバ X やその他のサーバ間でのネゴシエーションも実施して構わないが、本ガイドラインでは対象としない。本書では、議論の単純化のために、クライアントとサーバ間にてサーバ側が提供する Web API についてのみ規定する方針とした。

今回スコープ外とした構成については、将来の ECHONET Lite Web API ガイドライン第２版策定時に対象とするか検討していく。

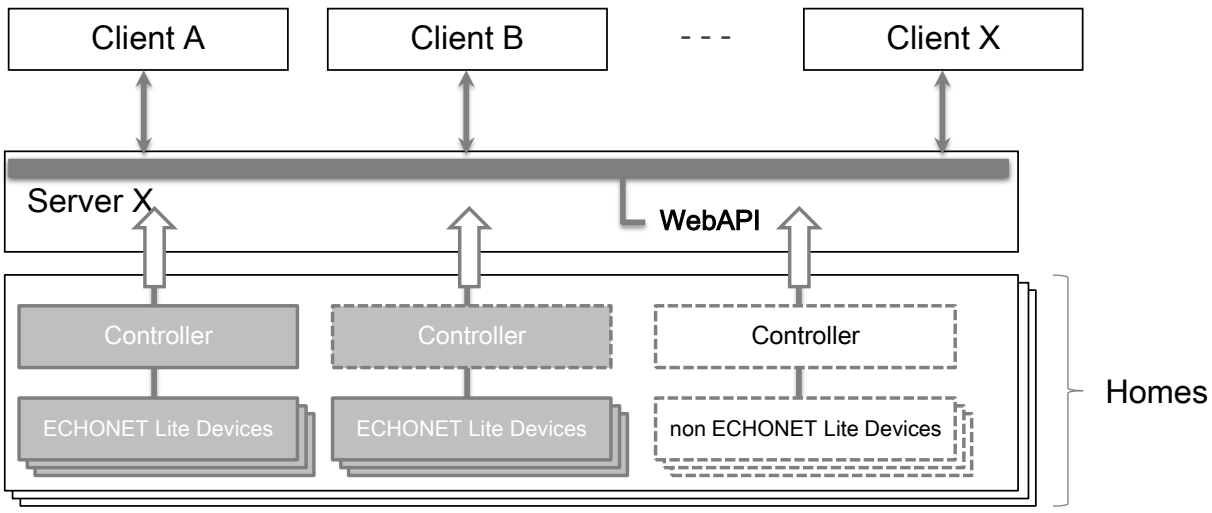


図 2-1　本ガイドライン対象のシステム構成図

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

第３章 ECHONET Lite Web API のユースケース

本章では、本ガイドライン策定において事前に検討されたユースケース候補についてまとめる。以降、各ユースケースを紹介し、３．７にて本書（第１版ガイドライン）で対象とする範囲について絞り込む。

３．１ 状態取得・制御・通知

ECHONET Lite の基本操作（ESV）である GET/SET/INF をマッピングするユースケース（図３-１）。GET については、宅内機器までの応答遅延を考慮し、サーバ内でキャッシングしたデータをクライアントへ返却するケースも含む。SET は、基本的に宅内機器への操作を想定する。INF のように、機器にて生じた状態変化通知などをできるだけ即座にサーバを介してクライアントへ伝えるケースも想定される。

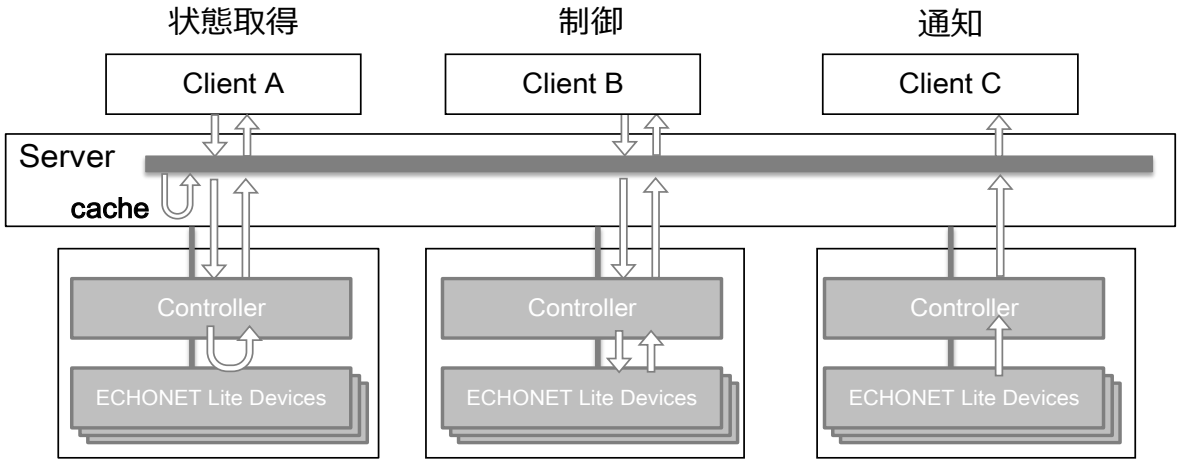


図 ３-１ ユースケース：状態取得・制御・通知

３．２ 機器の一覧取得・管理情報取得

機器の一覧（リスト）を取得する（図 ３-２の左）ユースケース。あるユーザ宅の機器全ての一覧を取得するケースや、エアコンのみといった機種指定による機器の一覧を取得するケースが考えられる。さらに、サーバが収容している全てのユーザが保有する機器全ての一覧や、その一部を切り取って扱うケースなどが考えられる。

また、ECHONET Lite コントローラは機器オブジェクトとしてコントローラクラスを搭載しており、オプション機能として同コントローラが管理対象とする機器のリストなど、機器管理情報を保持することが可能となっている。こうした機器管理情報をサーバ側へマッピングし、クライアントから取得可能とするケースが考えられる。機器管理情報は、サーバ上でキャッシュし、機器追加・削除などの変更時のみ通知によりサーバ内にキャッシュした機器管理情報を更新するケースも想定される（図 ３-２の右）。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

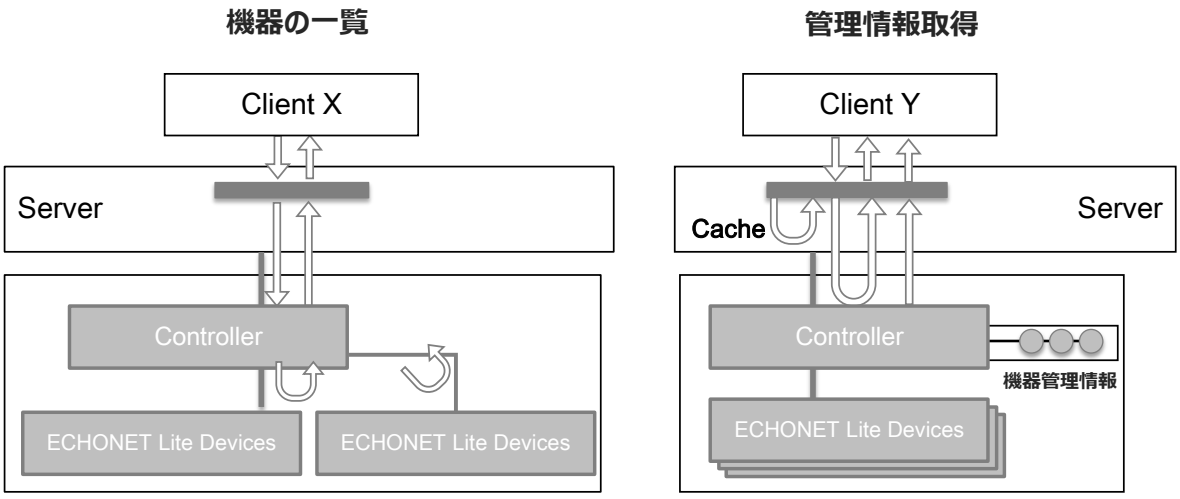


図 3-2 ユースケース：機器の一覧取得・管理情報取得

３．３ 機器の一括操作

ECHONET Lite では、コントローラ配下の全機器への一括操作や、エアコン全てといった機種指定による一括操作がサポートされている。これらをクライアントから操作可能とするユースケース。一人のユーザだけでなく、複数のユーザ宅内の機器を一斉に操作するケースも考えられる。

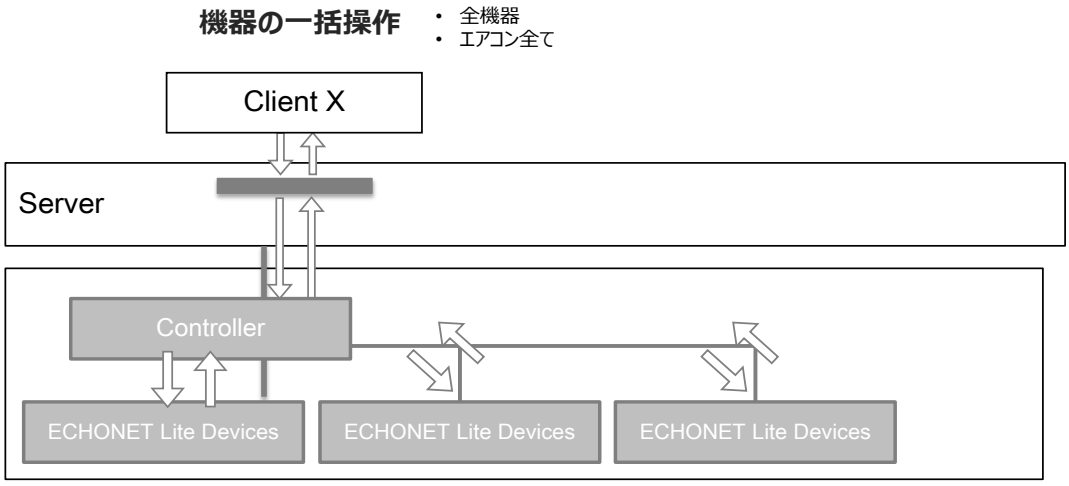


図 3-3 ユースケース：一括操作

３．４ 仮想機器

宅内の機器をサーバ上へマッピングする際、様々な情報加工を施すことで、見かけ上、機器の機能を拡張することが可能なユースケース。仮想機器として見せかけるバリエーションとしてここでは３種の想定例について列挙する（図 3-4）。

例えば、ある機器オブジェクト仕様 Appendix Release X 対応のエアコンをサーバへマッピングす

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

る際、Release X 時から最新の Release Y においても規格書の該当内容に関するプロパティについて変更が無い場合は、Release Y 対応エアコンとしてクライアントへ見せかけることが可能となる。旧モデル対応エアコンであっても、サーバ上での適切な変換処理により、新モデル対応エアコンと同様に扱えるなら、新モデルにしか対応しないクライアント提供アプリにも対応が可能となるメリットがある（図左）。

また、宅内のエアコンとなんらかのセンサを組み合わせて、新センサ付仮想エアコンとみなして、クライアントのアプリへ提供することも考えられる。エアコンとセンサを個別の機器として扱っても良いが、一体化することによりアプリの幅が広がる可能性がある（図中央）。

さらに、エアコン自体が保持しない機能であっても、サーバ上で付加機能を加えることで拡張エアコンとしてクライアントへ公開するケースも考えられる。天気予報付きエアコンのようなものがあっても良いかもしれない（図右）。

仮想機器を扱う事例のバリエーションとして、非 ECHONET Lite 機器を扱うことも考えられる。

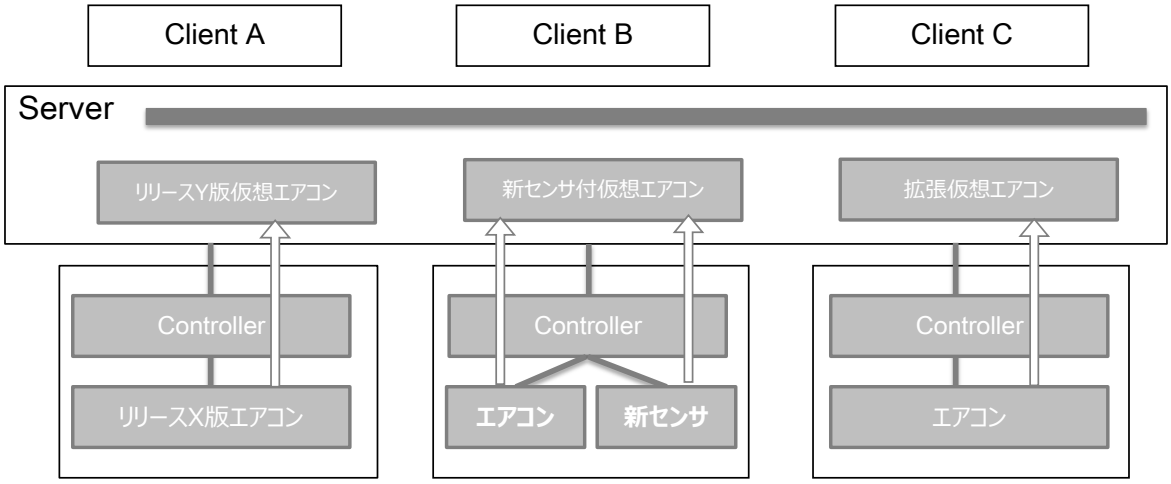


図 3-4　ユースケース：仮想機器

３．５ EMS 情報、サーバ蓄積ログ

宅内の機器から、各機器が保有するエネルギー関連情報（例えば、発電能力、蓄電能力、負荷情報など）を収集し、サーバ内で集約し EMS 情報としてクライアントへ提供することも考えられる。

また、宅内機器やセンサから得られる情報をサーバ上にて蓄積ログとして集積し、生活行動分析などに繋げるサービスも考えられる。

これらは、直接宅内 ECHONET Lite 機器をマッピングしたものではなく、機器から収集される情報を集約・加工したものをクライアントへ提供する形式となる（図 3-5）。

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

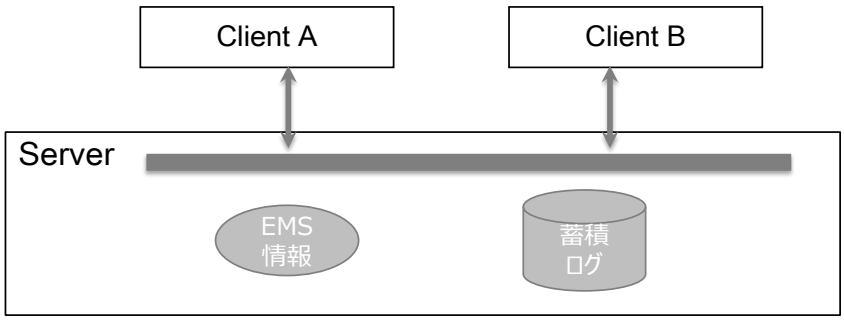


図 3-5 ユースケース：EMS 情報、サーバ蓄積ログ

３．６ 認証・認可

今回規定するような顧客情報を扱うケースでは、そもそもクライアントがサーバから Web API によるサービス提供を受けるにあたり、なんらかの認証や認可を得る必要があるのが一般的である。具体的には、OAuth2.0 [OAUTH]などのプロトコルを用いてセッションを確立した上で、サーバは、クライアント毎にアクセス許諾するリソース集合を準備し、実際にサービス提供することになる。

また、サーバ上では、顧客（ユーザ）情報の追加／削除や、顧客保有のコントローラや機器の追加／削除を実施可能な仕組みが必要となる。こうしたユーザ管理、機器管理機能を Web API にて外部提供することも考えられるが、サーバとユーザ間であらかじめ必要な作業を行う形式も考えられる（図 3-6）。

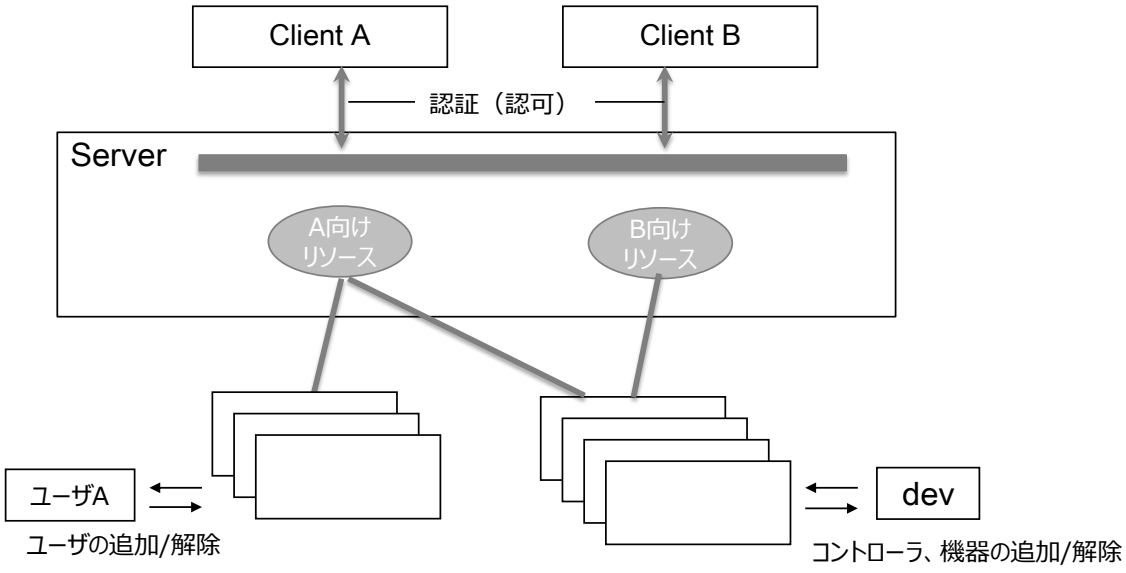


図 3-6 ユースケース：認証・認可

３．７ 対象ユースケースの絞り込み

本書の第１版ガイドラインは、早期にリリースすることを目標に議論が開始された経緯から、対象とすべきユースケースについて絞込みを実施する（表 3-1）。基本的には、ECHONET Lite 通信の

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

モデルを直接サーバ上へ展開･変換するユースケースを主対象とし､仮想機器など単純にECHONET Lite 通信にマッピングしづらいユースケースは、今回スコープ外とした。クライアント認証・認可については、直接対象としないが、事例については紹介する。

次期モデルについては､第 2 版ガイドライン (2018 年度末リリース予定) にて検討していく予定。

表 3-1 各ユースケースの本ガイドライン対応方針

ユースケース	本書での対応方針
機器単体の状態取得・制御・状変通知	基本モデルとして対応 一部、事例紹介
機器一覧取得	
コントローラ管理情報の取得・状変通知	
複数コントローラ対応	
機器一括制御	次期モデルとして今後検討
仮想機器対応	
EMS 情報取得・制御	
サーバ蓄積ログ取得	
認証・認可	事例紹介

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

第４章 Web API モデルの指針

４．１ 基本方針

本ガイドラインでは、Web API のモデルとして、REST（REpresentational State Transfer）を採用する。REST は、URI に対して HTTP [HTTP]メソッドをベースにシンプルな制御が可能であり、現在実質的な Web API の標準になっている。データ形式には現在主流である JSON [JSON]を採用し、Rest(ful) API となるよう全体設計する。

REST の４原則である、①HTTP によるステートレス性、②URI によるアドレス可視性、③HTTP メソッドによる統一インタフェース、④JSON によるリソース間の接続性に留意しつつ、API を規定する。

リソースの識別方法は、下記を基本とする。

形式	スキーム://ホスト名/リソースへのパス
例	https://www.example.com/elapi/v1/devices/12345678

この例では、HTTPS スキームにて、ホスト www.example.com が保有する ID が 12345678 の機器データのリソースを指定している（ポート番号については省略）。このように、パスで階層化してリソースを指定する REST API を基本モデルとする。また、クエリーパラメータでリソースを指定するクエリーAPI も補完的に選択可能とする。具体的なリソースの指定方法については、次節以降記述する。

リソースに対するアクションの多くは、CRUD（Create、Read、Update、Delete）で表現される作成、参照、更新、削除の各操作を参考に、リソースの URI に対する HTTP メソッドや URI のクエリーパラメータにて実装されるモデルを想定する。

前述以外の基本方針として、下記を定める。

- HTTP のバージョンは HTTP/1.1[HTTP]を採用する。
- JSON 形式は、RFC 8259 に従う。Content-Type として「application/json」を指定し、文字コードは UTF-8 を使用する。クライアント（クライアント）側では、Accept というリクエストヘッダでメディアタイプ「application/json」を指定する。
- スキーマには、HTTPS を使用する。
- 日時形式には、RFC 3339（ISO 8601）を採用し、タイムゾーンを考慮した下記表記に従う。
  2018-01-02T12:34:56+09:00
  ただし、年月日、時、分、秒など個別に指定すべきケースは個々の記述に従う。
- リソースに指定されるプロパティ名称など属性名の命名規則として、単一語の場合は小文字を使用するが、複合語の場合は（ローワー）キャメルケースを使用する。本ガイドラインでは説明の都合、任意の値や記述を表現する場合には山括弧「<>」を使用する。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

- ECHONET Lite の機器オブジェクトの Device Description 形式（後述）は、W3C で策定中の WoT モデルを一部参考として記述する。

次節以降、下記のリソース指定・操作例について記述する。

- アプリケーション名
- API バージョン、バージョン一覧取得
- 対象リソース種一覧取得
- 機器の一覧取得
- 機器情報（Device Description）取得
- 機器オブジェクトのプロパティ値操作（SET/GET など）
- キャッシュの扱い
- 機器オブジェクトのプロパティ値通知（INF）
- 認証・認可

以下で示す事例では、HTTP(S)リクエストでは、リクエストライン、リクエストヘッダ、メッセージボディ、HTTP(S)レスポンスでは、ステータスライン、レスポンスヘッダ、メッセージボディの内容のうち、（リクエスト／レスポンス）ヘッダ部については特別明記する必要がある場合を除き例示を省略する。

４．２ アプリケーション名

本書にて規定する ECHONET Lite Web API のアプリ名称を「elapi」と呼称する。トップ階層に「/elapi」というアプリケーション名をパスとして埋め込むことで、明確に ECHONET Lite Web API のコンポーネントであることを示す。

特定パス収容方式を選定した理由

API の指定方式として、下記のような記述例が候補となる。

① 特定のパスに収容する場合　`https://www.service.xx/elapi/xxx`

② ホスト名で区別する場合　　`https://elapi.service.xx/xxx`

Web アプリケーションの実装形態やサービス規模／負荷分散も考慮すべきだが、リバースプロキシなどを用いれば接続先 Web アプリケーションサーバの切り替えは可能なため、①特定パスに収容する方式に統一する。

４．３ API バージョン

本ガイドラインで定める API バージョンはリビジョン番号形式で管理するものとし、現時点では v1 とする。今後、細かな仕様追加や修正などを実施する場合であっても、後方互換性がとれる範囲で規定する場合は、バージョンを上げずに運用するものとする。将来、互換性の無い仕様変更が必要となった場合は、仕様を区別するため、v2 を策定する方針とする。なお、バージョン指定は、URI にバージョン情報を埋め込む方法を採用する。

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

URI 指定によるバージョン管理を選択した理由

バージョン指定方法には、下記のように大きく 3 通りの方法がある。
① URI に含める： https://www.service.xx/elapi/v1
②クエリーパラメータに含める：https://www.service.xx/elapi/xx/80234512?v=1.5
③HTTP ヘッダに含める：Accept: application/vnd.echonet.v2+json

②はバージョン情報が省略可能というメリットがあるが、デフォルトバージョンが最新版とすると、API 変更時にトラブルを発生しやすい。また、③はベンダ固有のメディアタイプとしてサーバへ要求し、対応可能な場合は「Content-Type: application/vnd.echonet.v2+json」と共に「Vary: Accept」をヘッダに付与して応答する必要があるが、Content-Type が application/json と一致しないと（独自のメディアタイプを）受け付けないライブラリが存在するケースもある。
本書では、識別しやすく、多くの Web API 提供サービスで広く使われていることを理由に、①を採用する。

4．4 バージョン一覧取得

トップの URI に対して「/elapi」を指定し、GET メソッドにて要求することで、サーバが対応するバージョン一覧を取得できる。なお、以降の例では、リクエストライン（と必要に応じてボディ）と、レスポンスボディのみを示す。リクエストラインの後の各種ヘッダ（Host ヘッダや Content-Type ヘッダなど）や、応答時のステータスライン、レスポンスヘッダについては記載を省略する。

■ リクエスト

```
GET /elapi HTTP/1.1
```

■ レスポンス

```
{
    "versions": [
        {
            "id": "v1",
            "status": "CURRENT",
            "updated": "2018-01-01T12:34:56Z+09:00"
        },
        {
            "id": "v2",
            "status": "EXPERIMENTAL",
            "updated": "2018-01-02T01:02:03Z+09:00"
        }
    ]
}
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

上記例では、v1 と v2 のふたつのバージョンをサポートしていることを示している。最低 1 つのバージョンはサポートすること。本書に対応したバージョンは v1 とし、将来のバージョン更新は v2 を予定する。開発レベルなどに応じて、status には、下記の状態を指定可能とする。

- CURRENT：最新の安定版
- SUPPORTED：安定版（最新版ではない）
- DEPRECATED：既に廃止となった版
- EXPERIMENTAL：開発・実験中の版

updated で、更新日時を指定可能とし、サーバ側システムに何らかの変更を実施したタイミングを記述可能とする。id は必須、status, updated はオプション指定とする。

４．５ 対象リソース種一覧取得

「/elapi/<versionId>」を指定し、GET メソッドにて要求することで、サーバが「/elapi/<versionId>」直下で対応するリソース種一覧を取得できる。

■ リクエスト

```
GET /elapi/v1 HTTP/1.1
```

■ レスポンス

```
{
    "v1": [
        {
            "name": "devices",
            "descriptions": {
                "ja": "devices の説明文",
                "en": "device resource"
            },
            "total": 10
        },
        {
            "name": "controllers",
            "descriptions": {
                "ja": "controllers の説明文",
                "en": "controller resource"
            },
            "total": 1
        },
        {
            "name": "sites",
            "descriptions": {
                "ja": "sites の説明文",
                "en": "sites resource"
            },
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
            "total": 5
        },
        {
            "name": "users",
            "descriptions": {
                "ja": "users の説明文",
                "en": "user resource"
            },
            "total": 100
        },
        {
            "name": "groups",
            "descriptions": {
                "ja": "groups の説明文",
                "en": "group resource"
            },
            "total": 3
        }
    ]
}
```


上記例では、devices, controllers, sites, users, groups の 5 種類のリソース種をサポートしていることを示している。本ガイドラインでは上記 5 種類を例示するが、同様のリソース種をサーバが任意に規定できるものとする。リソース種は、必ず 1 種類以上提供すること。クライアントに対して提供しないリソース種は記載しないこと。各リソース種の説明文を descriptions（必須）にて、各リソース種の総数を total（オプション）で示す。説明文の記述は任意の文字列とし、総数は整数値とする。基本的に、クライアント（提供先）の要求・権限や提供サービス内容などに応じて、各クライアントへ提供するリソース種を変更して（切り替えて）構わない。

devices 以外のリソース種を提供する場合、そのリソース種の配下のパスには devices を配置すること（詳細は後述）。

あるリソース種を指定する場合、サーバが保有する全てのリソース種集合（devices であれば全ての機器集合）を公開する、もしくは、クライアント毎に許可されたリソース種サブ集合（devices であれば機器集合セット）を公開するものとする。

サーバがクライアントに対して提供するリソース集合の範囲は、サーバ側で任意に規定できる。なお、devices を用いた具体的な応答事例については、次節で紹介する。

リソース種に controllers を指定する場合、サーバが保有する機器のうち、コントローラクラスに属する機器の一覧を公開する。また、「/controllers/<controller_id>/devices」と指定することで、controller_id にて指定するコントローラ配下に接続された機器一覧を指定可能とする。

図 4-1 では、サーバがクライアントへ提供するリソースの考え方について例示している。あるクライアントに対して公開可能な全ての機器一覧を「/elapi/v1/devices」で示している。機器数が多数となる場合、「/elapi/v1/devices?offset=0&limit=3」のように、範囲指定により機器一覧を取得できる機能も提供可能とする（オプション）。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

サーバがリソース種として users を提供する場合、例えば、S さん、T さんのようにユーザ指定した上で機器一覧を取得するには、「/elapi/v1/users/S/devices」や「/elapi/v1/users/T/devices」のように個々のユーザが保有する機器の一覧を区別して指定可能にする。

図 4-1 ではグループ A（例ではエアコンのグループ）を指定して機器一覧を取得する場合、「/elapi/v1/groups/A/devices」にて指定可能とする。

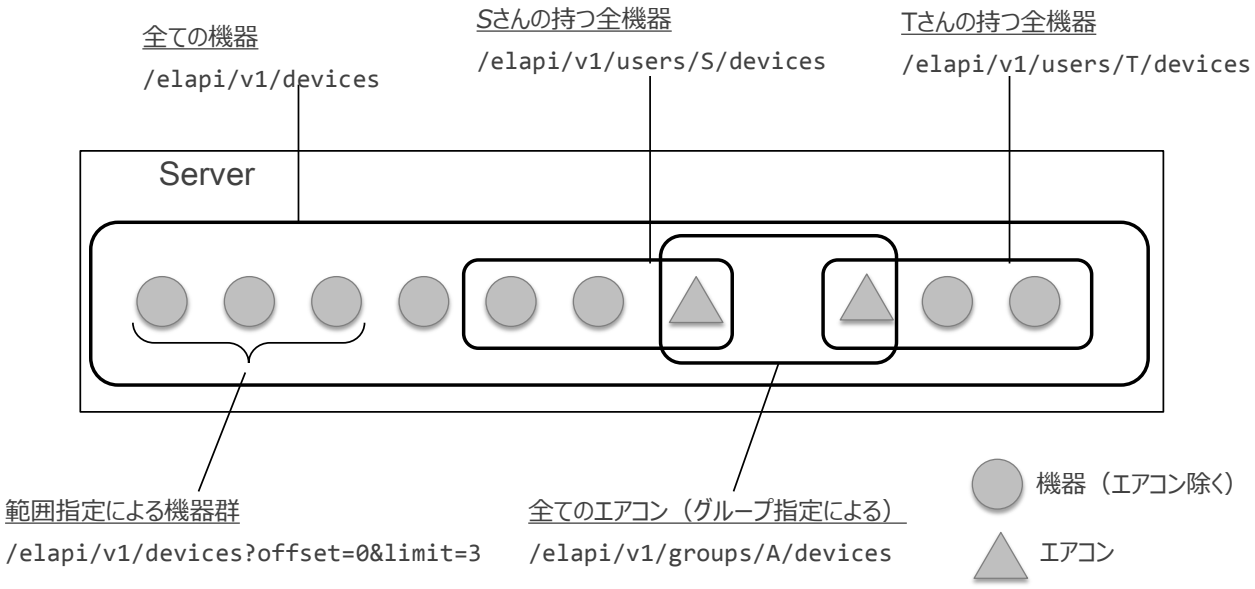


図 4-1 サーバが提供するリソース種の考え方

サーバがリソース種として sites を提供する場合、集合住宅、テナント、地域など、ユーザを集合化したい単位にて任意に扱うことが可能とする。

下記例のように、「/sites」にてサイト一覧、「/sites/<site_id>」にてサイトの description、「/sites/<site_id>/devices」にて site_id にて指定されるサイト内の機器一覧を指定することができる。ここでは、サイトに係る内容については特に説明しない。

■ リクエスト

```
GET /elapi/v1/sites HTTP/1.1
```

■ レスポンス

```
{
    "sites": [
        {
            "id": "site_id1",
            "descriptions": {"ja": "Aアパート", "en": "A apartment"},
            …
        },
        {
            "id": "…",
        }
    ]
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
}
```

■ リクエスト
```
GET /elapi/v1/sites/<site_id> HTTP/1.1
```

■ レスポンス
```
{
  <サイトの Site Description>
}
```

■ リクエスト
```
GET /elapi/v1/sites/<site_id>/devices HTTP/1.1
```

■ レスポンス
```
{
  <デバイス一覧>
}
```

以降、本ガイドラインでは devices のケースのみ規定する。他のリソース種については任意に規定可能とし、本書ではスコープ外とする。

4．6 機器一覧取得

「/elapi/v1/<リソース種（省略可）>/devices」 を指定して GET することで機器一覧を取得できる。

■ リクエスト
```
GET /elapi/v1/devices HTTP/1.1
```

■ レスポンス
```
{
   "devices": [
      {
         "id": "0xFE000006123456789ABCDEF123456789AB",
         "deviceType": "generalLighting",
         "protocol": {
            "type": "ECHONET_Lite v1.13",
            "version": "Rel.J"
         },
         "manufacturer": {
            "code": "<manufacturerCode>",
            "descriptions": {
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
                "ja": "<manufacturerName(日本語)>",
                "en": "<manufacturerName(English)>"
            }
        }
    },
    {
        "id": "",
    }
]
}
```

表 4-1 機器一覧取得時の応答内容

Property	Type	Required	Description	Example
id	string	Yes	機器固有の ID	"0xFE000006…AB"
deviceType	string	Yes	機器の種類	"generalLighting"
protocol	object	Yes	使用プロトコル	―
protocol.type	string	Yes	ECHONET Lite バージョン番号	"ECHONET_Lite v1.13"
protocol.version	string	Yes	Appendix リリース番号	"Rel.J"
manufacturer	object	Yes	メーカ情報	―
manufacturer.code	string	Yes	メーカコード	"0x******"
manufacturer.descriptions	object	Yes	メーカ名称	―
manufacturer.descriptions.ja	string	Yes	名称（日本語）	"A ベンダ"
manufacturer.descriptions.en	string	Yes	名称（英語）	"Vendor Name A"

id は、機器固有の ID であり、システム内（/devices 配下）にて重複しないユニークな値である必要がある。値の例としては、機器自体が保持する固有情報（MAC アドレスや機器オブジェクトの固有 ID など）や、コントローラが割り当てる値、サーバ内にて固有に割り当てる値などが想定される。上記例では先頭に”0x”を付加した 16 進数の文字列にて記載しているが、”0x”の無い任意の文字列で構わない。

deviceType は、機器オブジェクトのクラス名に相当する名称となる。具体的に使用可能なクラス名称については、本 Web API Appendix「4.2 機器毎の Device Description」部にて記載する。

protocol は、type で ECHONET Lite のバージョン番号を、version で機器オブジェクトの Appendix リリース番号（EPC=0x82）を ASCII 文字列にて記載する。

manufacturer は、機器オブジェクト・スーパークラスのメーカコード（EPC=0x8A）のプロパティ値を code に、description にて日本語(ja)、英語(en)での名称を記載する。

機器一覧は多数となる場合がある。サーバが一度にクライアントへ返却する機器数を制限したい場合には、下記のように、後続機器の有無（"hasMore"）、一度に返却する数（"limit"）、先頭からのオフセット位置（"offset"）を合わせて返却しても良い。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
{
    "devices": [
        :
    ],
    "hasMore": true,
    "limit": 25,
    "offset": 0
}
```

また、クライアント側からオフセット位置と返却数を指定するクエリー形式もサポートしても良い。

```
GET /elapi/v1/devices?offset=0&limit=25 HTTP/1.1
```

4．7 機器情報（Device Description）取得

「/elapi/v1/<リソース種（省略可）>/devices/<deviceId>」 を指定して GET することで、機器に対応する仕様を JSON 形式の 記述「Device Description」として取得可能とする。Device Description は、機器が実装している機能（properties, actions, events）の定義である。

各機器の Device Description は本 Web API Appendix の「4. Device Description」に記述されている。

■ リクエスト

```
GET /elapi/v1/devices/<deviceId> HTTP/1.1
```

■ レスポンス

```
{
    "deviceType":<device type>,
    "eoj":<eoj in Hex string>,
    "descriptions": {
        "ja": <description of property in Japanese>,
        "en": <description of property in English>
    },
    "properties": { <property1>: <property object>, <property2>: <property object> ...
    },
    "actions": [ <action object>, <action object>...
    ],
    "events": [ <event object>, <event object>...
    ]
}
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

表 4-2 機器情報取得時の応答内容

Property	Type	Required	Description	Example
deviceType	string	Yes	device の種類を示す。ECHONET Lite の機器オブジェクト名(EOJ)に対応する。値に関しては "Device Types" を参照のこと。	"generalLighting"
eoj	string	No	ECHONET Lite の EOJ のクラスコードを string で Hex 表記（ID として使用しない）	"0x0130"
descriptions	object	Yes	ECHONET Lite で定義された機器オブジェクトの名称	
descriptions.ja	string	Yes	ECHONET Lite の機器オブジェクト	"一般照明"
descriptions.en	string	Yes	device object name of ECHONET Lite in English	"General Lighting"
properties	object	Yes	property object の列挙。property1, property2 は、property resource name（本 Web API Appendix 記載）	
actions	array	No	action object の array	
events	array	No	event object の array	

Property object

Property object は機器の Property を記述する。ECHONET Lite で GET または SET をサポートするエコーネットプロパティに対応する。EL-Web API で property を取得するには GET メソッド, 設定するには PUT メソッドを使用する。property object の構成を以下に示す。

```
{
    "epc":<epc>,
    "epcAtomic":<epc>,
    "descriptions": {
        "ja": <description of property in Japanese>,
        "en": <description of property in English>
    },
    "writable":<writable flag>,
    "observable":<observable flag>,
    "querySchema": <schema of parameters>,
    "schema":<schema of data>,
    "note": {
        "ja":<note in Japanese>,
        "en":<note in English>
    }
}
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

表 4-3 Property Object の内容

Property	Type	Required	Description	Example
epc	string	No	対応する ECHONET Lite の EPC を string で Hex 表記する	"0x80"
epcAtomic	string	No	ECHONET Lite として Atomic operation が必要な EPC	
descriptions	object	Yes	ECHONET Lite で定義されたエコーネットプロパティの名称	
descriptions.ja	string	Yes	ECHONET Lite のプロパティ名	"動作状態"
descriptions.en	string	Yes	property name of ECHONET Lite in English	"Operation Status"
writable	boolean	Yes	書き込みが可能か不可能かを示す。ECHONET Lite の SET に対応	true, false
observable	boolean	Yes	状態変化を通知できるか否かを示す。ECHONET Lite の INF に対応	true, false
querySchema	object	No	GET で parameter を渡す必要がある場合に query を利用する。parameter の情報を JSON Schema 形式で記述	
schema	object	Yes	property data の情報を JSON Schema 形式で記述	
note	object	No	Property に関する付加情報	
note.ja	string	No	付加情報を日本語で記述する	
note.en	string	No	付加情報を英語で記述する	

Action object

本書（第１版ガイドライン）では具体的な記述例を示さない。第２版ガイドラインにて追記を予定している。

Event object

本書（第１版ガイドライン）では具体的な記述例を示さない。第２版ガイドラインにて追記を予定している。

４．８ 機器オブジェクトのプロパティ値操作（SET/GET など）

エコーネットでは、プロパティに対する操作を ESV（ECHONET Lite サービス）として規定している。エコーネットにおけるプロパティ値をサーバ上へマッピングしてリソースとして公開し、ESV を HTTP メソッド (CRUD 操作) に対応付けて操作することで、Restful な API 操作が可能となる。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

機器オブジェクトのプロパティレベルの具体的なマッピングおよび操作については、次章にて議論する。

４．９ クライアントのキャッシュの扱い

クライアントからサーバを経由して宅内機器を操作する（状態取得や制御を実施する）場合、処理によっては、応答まで長い時間を要するケースがある。HTTP キャッシュ（RFC 7234）は必須ではないが、クライアントからサーバに対して値を取得するケースにおいて、キャッシュしたリソースの再利用は有効となるため、参考として例示する。

メーカコードなどの静的データ（基本的に長期不変・固定情報）や、前回アクセスから一定期間データ更新が行われないデータ（動的情報）を扱う場合などにおける HTTP キャッシュの活用例について以下に示す。

HTTP キャッシュのモデルには、Validation Model（検証モデル）と Expiration Model（期限切れモデル）がある。

Validation Model

データ更新されていた場合のみ取得するモデルとなる。

- サーバ：最終更新日付かエンティティタグを必要に応じて使用する

```
Last-Modified: True, 02 Jan 2018 00:00:00 GMT
ETag: "fa39b31e285573ee37af0d492aca581"
```

- クライアント：最終更新日付にて条件付きリクエストを用いる時

```
GET /elapi/v1/devices/12345 HTTP/1.1
If-Modified-Since: True, 02 Jan 2018 00:00:00 GMT
```

- クライアント：エンティティタグにて条件付きリクエストを用いる時

```
GET /elapi/v1/devices/12345 HTTP/1.1
If-None-Match: "fa39b31e285573ee37af0d492aca581"
```

- サーバ：変更なければステータスコード 304（と空ボディ）、あれば 200（と変更内容）を返信

Expiration Model

キャッシュの有効期限切れ時のみ取得するモデルとなる。

- サーバ：期限切れ日時か現在時刻からの秒数を指定する。

```
Expires: Wed, 03 Jan 2018 00:00:00 GMT
```

```
Cache-Control: max-age=3600
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

前者は、指定された期限まではキャッシュデータを利用し応答する。
後者は、リアルタイム性が低いケースであり、Date ヘッダからの経過秒数を指定する。
両方使う場合は、Cache-Control 優先となる。
HTTP ヘッダ内の日時形式（RFC 7231 (HTTP1.1)の HTTP-data 記載の 3 種）に対応すること。

- クライアント：上記レスポンスヘッダの値を参考に、次回リクエストの発行タイミングを考える。

上記はキャッシュの有効活用を想定したが、逆に、キャッシュを全く使用しない（させない）ケースも存在する。

（明示的に）キャッシュさせたくないケース

Cache-Control ヘッダを用いることにより、クライアントに毎回アクセスを要求する（クライアントにキャッシュや再利用しないように指示する）ことができる。

- サーバ：検証が必要（現在でも有効か否かをサーバに問い合わせ、確認がとれない限り再利用してはならない）と明示

```
Cache-Control: no-cache
```

- サーバ：キャッシュをしてはならないことを明示

```
Cache-Control: no-store
```

４．１０ 機器オブジェクトのプロパティ値通知（INF）

本ガイドラインで想定する Web API では、基本的にクライアントからの要求に即座に返答する通常の同期的な HTTP REST を想定しているため、機器側のタイミングで非同期的に情報伝達を行う INF を利用するには工夫が必要である。

INF の取扱いについて

最も単純には、サーバ側において INF を受信した際にその値を用いてキャッシュを更新しておき、サービサーからはそれに定期的にアクセスする（ポーリングを行う）ことでその更新を検知する方法が考えられる。このやりとりには４．９にて説明した Validation Model を用いることができる。この手法は非常に簡便だがいくつかの問題がある。一つには情報伝達タイミングがクライアントからのリクエストのタイミングに限られるために、サーバが非同期的に INF を受け取った瞬間にその値をクライアントに伝達することができず常に遅延が生じるという点である。もう一つには、クライアントがその値に関心があるかどうかにかかわらず、サーバでは INF を受け取りうるプロパティ全てに対して履歴を保存する必要があるという点である。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

従って、INF を適切に処理するにはサーバ側からクライアントへメッセージを送信するプッシュ通知型の通信方式についてもサポートされることが望ましい。サーバからプッシュ通知を行う手法はいくつか知られている。

- W3C 関連：
  - Push API（Web Push。単一ブラウザ通知。サーバから Push 可能。Android 対応、iOS 非対応）
  - Service Workers
  - Web Notification（単一ブラウザ通知。Android 対応、iOS 非対応）
- スマホアプリ関連（ネイティブ Push）：
  - APNS（Apple Push Notification Service）
  - GCM（Google Cloud Messaging）、FCM（Firebase Cloud Messaging）
- その他：
  - WebSocket
  - MQTT

また、REST で提供される手段を用いながらもコネクションを比較的長時間持続させるロングポーリングと呼ばれる方法もある。
本節では、ロングポーリングと WebSocket、そして MQTT による実現事例について紹介する。

ロングポーリングによる INF 実現例

本来 HTTP REST でのアクセスは、リクエストを受けたら即座にレスポンスを返すことが期待されている。「ロングポーリング」とはこのレスポンスを遅らせ、その間サーバとクライアントの間にコネクションを維持するという手法である。これにより、サーバ側で何か情報が発生した段階で即座にそれを送信することができる。通常の HTTP アクセス手段を流用して実装できる汎用性の高い方法であるが、サーバとの同時接続数が増える傾向にあるため、その対策を講じることや、切断時の再接続をうまくハンドリングする必要性が生じる。

第５章で詳細説明される GET メソッドによるプロパティ値の取得方法を踏まえ、INF の受信タイミングをロングポーリングで受け取ることを考える。このリクエストを通常の GET と区別するため、INF 向けには他のメソッド、例えば POST で取得することで、サーバ側にロングポーリングをリクエストすることとする。サーバ側は、値が更新されるまではレスポンスを返さないように実装する。

■ リクエスト

```
POST /elapi/v1/devices/<deviceId>/properties/<PropertyResourceName> HTTP/1.1
```

■ レスポンス（更新されるまで返答しない。内容は GET の場合と同一である）

```
{
    <propertyResourceName>:<data>
}
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
or

■ レスポンスが object の場合
{
    <propertyResourceName>: {
        <elementName>:<data> ,
        <elementName>:<data> ,
        ...
    }
}
```

さらに、なんらかの要因により接続が切れて再接続するまでの間にサーバが INF を受け取ってしまった時にも、その情報を再接続時に受け取ることを可能にすることが望ましい。これにはリクエストヘッダに If-Modified-Since ヘッダを付与し、その時点以降に変更が生じているならばサーバは即座にレスポンスを返すように実装すればよい。この部分はサーバ側に値を記録しておかねばならないという点で冒頭に述べたキャッシュによる実装に似通っているが、① ロングポーリングでの INF 取得を仮定する場合は切断から再接続までの時間はさほど長くないと想定できるため、履歴を保存しておくべき時間を長くする必要性は低いこと、 ② POST でアクセスされていないプロパティの値を保存する必要はないなどの点から、依然単純なキャッシュ実装より優れている。

WebSocket による INF 実現例

WebSocket をはじめコネクションベースの方法を用いる場合は、通信モデルとして Pub/Sub を用いると簡便である。Pub/Sub ではトピックと呼ばれる文字列を介して通信を行う。このトピックとしてこれまで説明してきたリソース名を用いることで、本 Web API の基本概念を大幅に変更することなく INF 向け拡張を実装することができる。

Pub/Sub モデルをプロトコルレベルでサポートしている場合（MQTT や WAMP など）では、メッセージの配送だけを専門に担当するブローカーやルーターと呼ばれるサーバを用いることが多いが、本節で説明する、WebSocket 上で実装する Pub/Sub では REST からの連続性に留意し、サーバがブローカー機能を兼ね、通信はサーバとクライアント間で起こるものとする。

リクエストとレスポンスの例（サブプロトコルとして「echonet」という文字列を採用している）

```
■ リクエスト
GET /websocket HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
Sec-WebSocket-Protocol: echonet
Sec-WebSocket-Version: 13

■ レスポンス
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: echonet

■(クライアント)サブスクリプション
{
    "method": "subscribe",
    "path": "/elapi/v1/devices/<device_id>/properties/operationStatus"
}
■(サーバ)サブスクリプション Ack
{
    "method": "subscribeAck",
    "path": "/elapi/v1/devices/<device_id>/properties/operationStatus",
}
■(サーバ)パブリケーション
{
    "method": "publish",
    "path": "/elapi/v1/devices/<device_id>/properties/operationStatus",
    "value": false
}
■(クライアント)アン・サブスクリプション
{
    "method": "unsubscribe",
    "path": "/elapi/v1/devices/<device_id>/properties/operationStatus",
}
■(サーバ)アン・サブスクリプション Ack
{
    "method": "unsubscribeAck",
    "path": "/elapi/v1/devices/<device_id>/properties/operationStatus",
}
```

MQTT による INF 実現例

MQTT は TCP 上で Pub/Sub モデルを実装した軽量プロトコルで、IoT に適した様々な特性を持っている。その詳細は省くが、これまで説明してきた方法との重要な差異としては、MQTT ネットワークの構成には、メッセージの生成と消費を行うクライアント以外にメッセージの配送を担当するブローカーが必要であるという点である。従って、本節で説明する INF の受け取りに MQTT を用いる場合にも、別にブローカーが必要となる。これまで説明してきた API サーバも、その API を利用

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

するクライアントも、MQTT ブローカーの観点からは MQTT クライアントであるが、API サーバの方は Publish のみを行う MQTT クライアント(Publisher)、クライアントの方は Subscribe のみを行う MQTT クライアント(Subscriber)となる。

なお、MQTT は TCP 上に直接実装されたバイナリのプロトコルであり、HTTP を用いていないため、正確には Web API ではない。Web クライアント、例えばブラウザから MQTT を利用したい場合は、AWS IoT や Mosquitto など、MQTT over WebSocket を行うことが可能なブローカーを用いる必要がある。

コンセプトは非常にシンプルで、前節で説明した WebSocket でも用いた Pub/Sub が素直に実装されるが、そのシーケンスは QoS と呼ばれる、メッセージの到達保証の度合いをコントロールするパラメータ、およびそれに伴う再送の回数により異なってくる。WebSocket による INF 実現例で提示したリクエストとレスポンスの例と最も近いのは QoS=1 のときであるので、同様のことを MQTT で実現した場合のシーケンス図を次に示す（バイナリプロトコルである MQTT では、リクエストラインのようにメッセージを平易に記述する手法があるわけではない）。

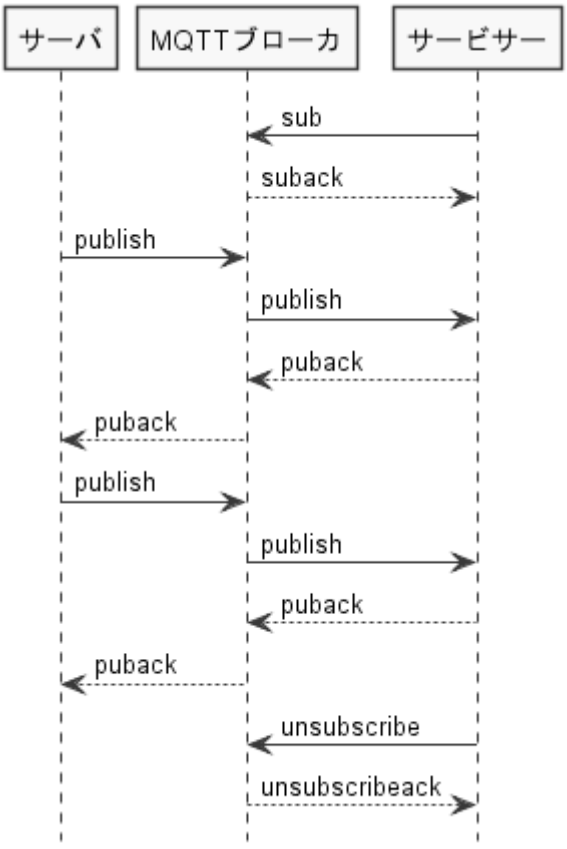


図 4-2 MQTT による INF 実現例

このように、ブローカーが挟まっている以外はほとんど WebSocket と同様である。Publish/Subscribe を行うトピックも、WebSocket のリクエストとレスポンスの例で提示した path の部分（REST で用いたリソース文字列そのもの）とし、ペイロードとして適宜 value を入れて実装すればよい。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

４．１１ 認証・認可

API 利用者の特定や、呼び出し回数のカウントにより過剰なアクセス時に通信遮断するなど、認証が必要となるケースは多いと考えられる。

本 Web API ガイドラインでは、認証・認可の仕組みについて参考事例として紹介する。Web API における認証・認可方式は、トークン認証や OAuth2.0 認可などが挙げられるが、どの方式を採用するかは、ユースケースによって異なりうる。

ここでは、OAuth2.0 を使用した例を示す。

Grant Type は 4 種（Authorization Code, Implicit, Resource Owner Password Credentials, Client Credentials）あり、用途によって自由に選択可能とする。

（下記は、Resource Owner Password Credentials の例）

OAuth のエンドポイントとして、「/elapi/v1/oauth2/token」を POST メソッドで呼び出すことで、ユーザ認証を実施する。リクエストメッセージボディに username や password の値を含み、上記 API リクエストに成功すると、ユーザには「トークン」の情報が返却される。

トークンは、以降の操作に必要な認証用文字列であり、認証以降のすべてのリクエスト時にこのトークンを付与する。

クライアントのリクエスト例：

```
POST /elapi/v1/oauth2/token HTTP/1.1
Host: xxx.com
Authorization: Basic ******
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=usr1&password=xx&scope=xx
```

サーバのレスポンス例：

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
    "access_token": "****",
    "token_type": "bearer",
    "expires_in": 360000,
    "refresh_token": "xxxx"
}
```

クライアントのリクエスト例（リクエストヘッダに入れる場合）：

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
GET /elapi/v1/devices HTTP/1.1
Host: xxx.com
Authorization: Bearer ****
```

クライアントのリクエスト例（リクエストボディに入れる場合）：

```
POST /elapi/v1/devices HTTP/1.1
Host: xxx.com
Content-Type: application/x-www-form-urlencoded

access_token=****
```

クライアントのリクエスト例（クエリーパラメータに入れる場合）：

```
GET /elapi/v1/devices?access_token=**** HTTP/1.1
Host: xxx.com
```

クライアントのリクエスト例（リフレッシュの場合）：

```
POST /elapi/v1/oauth2/token HTTP/1.1
Host: xxx.com
Authorization: Basic ******
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=xxxx
```

サーバのレスポンス例（有効期限切れ）：

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
    "error": "invalid_token"
}
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

第５章 ECHONET Lite 仕様のマッピング指針

本章では ECHONET Lite のフレームや機器オブジェクトの各仕様をどのようにマッピングすべきか、その指針について示す。

５．１ ECHONET Lite フレームのマッピング

ECHONET Lite は、下位通信層非依存の通信プロトコルである。一般的に、IPv4 や IPv6 使用時にはトランスポート層に UDP（必須）、TCP（オプション）を用いて、ECHONET Lite フレーム（表 5-1）をペイロードに搭載して通信を行う。

表 5-1　ECHONET Lite フレームの対応

フィールド	説明	Web API での対応
EHD	ECHONET Lite ヘッダ	非サポート
TID	Transaction ID	非サポート
SEOJ	Source ECHONET オブジェクト	非サポート
DEOJ	Destination ECHONET オブジェクト	機器オブジェクト（インスタンス）の id にて識別
ESV	ECHONET Lite サービス	後述
OPC	処理対象プロパティカウンタ	非サポート
EPC	ECHONET プロパティ	後述
PDC	プロパティデータカウンタ	非サポート
EDT	ECHONET プロパティ値データ	後述

表 5-1 に、ECHONET Lite フレームを構成する各フィールドに関する本 Web API での対応指針について記載する。主に、DEOJ（相手先機器オブジェクト）と ESV（サービス種）、EPC（プロパティ）、EDT（プロパティ値データ）を対象にマッピングすることで、Web API への変換を実現する。EHD や TID、PDC は宅内のコントローラと機器間でのみ関連する事項とし、サーバから外部へは公開されない情報とする。TID は、コントローラと機器間の通信管理に使用され、クライアントとサーバ間のセッション管理の一部として扱うことも考えられる。しかしながら、クライアントからのリクエストに対して、サーバ側がコントローラや機器に対して再送や複合電文を含む複数の形式で通信を実施するケースも考えられ、サーバ実装に依存することから、TID は本 Web API へのマッピングのスコープ外とする。

５．２ DEOJ のマッピング

ECHONET Lite のオブジェクトは、ノードプロファイルオブジェクトと機器オブジェクトの二種類が存在するが、機器オブジェクトのみ扱う。これは、Web API を介して操作を行うクライアント側アプリケーションは通常、機器オブジェクトへの操作が主体となるためである。通信部の確認など

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

ノードプロファイルオブジェクトへの操作が必要になるケースも考えられるが、第 1 版ガイドラインではスコープ外とする。

コントローラは機器に含まれるとする。

下記では、複数の機器を指定する場合の記述例について示し、一覧取得に使用するものとする。なお、一括制御に適用する複数機器の指定方法については、第 2 版ガイドラインにて検討する。

コントローラすべてを指定したい場合：

全コントローラを指定したい場合は、後述の特定機種すべてを指定したい場合の方法に従い type=controller とする、もしくは、サーバが controller というリソース種をサポートする場合は「/elapi/v1/<リソース種（省略可）>/controllers」と指定することができる。

機器すべてを指定したい場合：

全機器（全コントローラを含む）を指定したい場合は、「/elapi/v1/<リソース種（省略可）>/devices」と指定することができる。

特定機種すべてを指定したい場合：

ある機種をすべて指定したい場合は、
「/elapi/v1/<リソース種（省略可）>/devices?type=air_conditioner」のように、機種のタイプを指定すればよい。

インスタンスコードが 0x00 の場合の扱い：

ECHONET Lite では、あるノード上のある機種オブジェクトに関して、インスタンスコードを 0x00 とすることで、同機器オブジェクトのインスタンスすべてを指定可能であるが、本 Web API へのマッピングは行わないものとする

５．３ ESV のマッピング

ECHONET Lite における ESV は、表 ５-２のように定義されている。

表 ５-２　ESV の対応

ESV	サービス内容	記号	Web API での対応
0x60	プロパティ値書き込み要求（応答不要）	SetI	PUT メソッド（リクエスト）
0x61	プロパティ値書き込み要求（応答要）	SetC	
0x62	プロパティ値読み出し要求	Get	GET メソッド（リクエスト）
0x63	プロパティ値通知要求	INF_REQ	後述
0x6E	プロパティ値書き込み・読み出し要求	SetGet	PUT メソッド（リクエスト）
0x71	プロパティ値書き込み応答	Set_Res	PUT メソッド（レスポンス）
0x72	プロパティ値読み出し応答	Get_Res	GET メソッド（レスポンス）

© 2018　ECHONET CONSORTIUM　ALL RIGHTS RESERVED

0x73	プロパティ値通知	INF	後述
0x74	プロパティ値通知（応答要）	INFC	後述
0x7A	プロパティ値通知応答	INFC_Res	後述
0x7E	プロパティ値書き込み・読み出し応答	SetGet_Res	PUT メソッド（レスポンス）
0x50	プロパティ値書き込み要求不可応答	SetI_SNA	PUT メソッド（レスポンス）
0x51	プロパティ値書き込み要求不可応答	SetC_SNA	PUT メソッド（レスポンス）
0x52	プロパティ値読み出し不可応答	Get_SNA	GET メソッド（レスポンス）
0x53	プロパティ値通知不可応答	INF_SNA	後述
0x5E	プロパティ値書き込み・読み出し不可応答	SetGet_SNA	PUT メソッド（レスポンス）

ESV の種類は、要求、応答、不可応答に大別されるが、Set 操作や Get 操作を HTTP メソッドへマッピングする場合、要求（リクエスト）と応答・不可応答（レスポンス成功・失敗）に対応付けることができる。

Set 系 ESV は、値の更新や制御に相当するため、PUT メソッドへ対応づける。SetI, SetC, SetGet の 3 種類は区別なく、応答にてプロパティ値も返却される形式に統一し、PUT メソッドによるリクエストとレスポンスを使用する。また、AIF 認証仕様にて規定されている機器の応答待ち時間に関連し、Ack の返却、タイムアウト処理などをどのようにマッピングするかは、第 1 版ガイドラインではスコープ外とする。

Get 系 ESV は、値の取得・読み出しであるため、GET メソッドへ対応づける。サーバ上にキャッシングされるデータの扱いについて後述する。

INF 系 ESV は、通常の Web API ではプル・ベースとなるため、リアルタイム通知を受けることは難しいが、擬似的に GET メソッドを用いてポーリングすることや、WebSocket、MQTT、Web Notification API などのプッシュ・ベース通信モデルを活用することで実現する。

５．４ EPC、EDT のマッピング

EPC、EDT は、機器オブジェクト仕様とリンクして定義づけられている。次節以降、詳細に見ていく。

５．５ ECHONET Lite 機器オブジェクトのマッピング方式および操作

ECHONET Lite では、二種類のオブジェクト（ノードプロファイルオブジェクトと機器オブジェクト）を定義しているが、本 Web API ガイドラインでは、前述の通り、機器オブジェクトのみを扱う。機器オブジェクトのスーパークラスは、基本的に必須プロパティをマッピングするが、プロパティマップなど、JSON 化により自明となる情報は省略される。

GET /<deviceId>/Properties

「/elapi/v1/<リソース種（省略可）>/devices/<deviceId>/properties」を指定し GET することで、GET 対象となるすべてのプロパティについて値を取得可能とする。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

■ リクエスト
```
GET /elapi/v1/devices/<deviceId>/properties HTTP/1.1
```

■ レスポンス
```
{
    <propertyResourceName>:<propertyValue>,
    <propertyResourceName>:<propertyValue>
    ...
}
```

一般照明の例

■ リクエスト
```
GET /elapi/v1/devices/<deviceId>/properties HTTP/1.1
```

■ レスポンス
```
{
    "operationStatus":true,
    "faultStatus":false,
    "brightness":50,
    "operationMode":"color",
    "rgb":{ "r":20, "g":255, "b":0 }
}
```

GET /<deviceId>/Properties/<propertyResourceName>?<Query>

「/elapi/v1/<リソース種（省略可）>/devices/<deviceId>/properties/<propertyResourceName> ?<Query>」を指定し GET することで、指定したプロパティ値を取得可能とする。Query は通常は不要だが、スマートメータなど一部機器の特定プロパティで SET(EL)と GET(EL)の atomic operation が必要な場合に SET の data を指定するために Query を利用する。key は property 毎に定義する

■ リクエスト
```
GET /elapi/v1/devices/<deviceId>/properties/<propertyResourceName>?<Query> HTTP/1.1
```

■ レスポンス
```
{
    <propertyResourceName>:<data>
}
```

or

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

■ レスポンスが object の場合

```
{
    <propertyResourceName>: {
        <elementName>:<data> ,
        <elementName>:<data> ,
        ...
    }
}
```

一般照明の例

■ リクエスト

```
GET /elapi/v1/devices/<deviceId>/properties/operationMode HTTP/1.1
```

■ レスポンス

```
{
    "operationMode": "color"
}
```

■ リクエスト

```
GET /elapi/v1/devices/<deviceId>/properties/rgb HTTP/1.1
```

■ レスポンスが object の場合

```
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

■ リクエストに query がある場合

```
GET
/elapi/v1/devices/<deviceId>/properties/normalDirectionIntegralElectricEnergyLog1?day=0
HTTP/1.1
```

■ レスポンス

```
{
    "normalDirectionIntegralElectricEnergyLog1": {
        "day": 0,
        "energy": [
            20,
            34,
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
            59,
            109
        ]
    }
}
```

PUT /<deviceId>/Properties/<propertyResourceName>

「/elapi/v1/<リソース種（省略可）>/devices/<deviceId>/properties/<propertyResourceName>」を指定し PUT することで、指定したプロパティ値を設定可能とする。

■ リクエスト

```
PUT /elapi/v1/devices/<deviceId>/properties/<propertyResourceName> HTTP/1.1
{
    <propertyResourceName>:<data>
}

or
```

■ リクエストが object の場合

```
{
    <propertyResourceName>: {
        <elementName>:<data> ,
        <elementName>:<data> ,
        ...
    }
}
```

■ レスポンス

リクエストと同様

一般照明の例

■ リクエスト

```
PUT /elapi/v1/devices/<deviceId>/properties/operationMode HTTP/1.1
{
    "operationMode": "color"
}
```

■ レスポンス

```
{
    "operationMode": "color"
}
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

■ リクエストが object の場合

```
PUT /elapi/v1/devices/<deviceId>/properties/rgb HTTP/1.1
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

■ レスポンス

```
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

上記手順は、本 Web API Appendix 記載の各機器を対象に、Device Description の内容に基づいた操作を実行することを想定している。詳細については Appendix を参照すること。

一方、Appendix で規定する Device Description 内に所望のプロパティ定義が存在しない場合や、ベンダ独自のメーカ依存コードを使用する場合など、本ガイドラインの規定範囲外となる EPC 操作を可能にする補完機能を導入する（オプション）。

POST /<deviceId>/echoCommands

「/elapi/v1/<リソース種（省略可）>/devices/<deviceId>/echoCommands」を指定し POST することで、指定したプロパティ値を設定可能とする。Appendix で規定する Device Description 内に所望のプロパティ定義が存在しない場合や、メーカ依存コードを使用する場合など、本書の規定範囲外となる EPC 操作を可能にする補完機能として導入する。

ESV、EPC、EDT コードは[EL]にて定義される 16 進数の文字列（例：”0x80”）や配列にて指定する（数字や 10 進数文字列などの使用は本ガイドラインでは対象外とする）。また、複合電文も指定可能とする。

POST リクエスト時に要求用 ESV コード、POST レスポンス時に応答・通知用 ESV コードや不可応答用 ESV コードを使用する。リクエストに応じて「epc」、「edt」を返し、レスポンスが不可応答用 ESV コードとなる場合は、「edt」に「null」を指定すること。機器へ到達しないエラーが発生する場合は５．６に従い、適切にエラーを返却すること。

下記では、要求用 ESV コードとして Get（ESV=0x62）と SetGet（ESV=0x6E）、応答用 ESV コードとして Get_Res（ESV=0x72）と SetGet_Res（ESV=0x7E）を対象とした例について示している。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

Get の例

■ リクエスト

```
POST /elapi/v1/devices/<deviceId>/echoCommands HTTP/1.1
{
    "request": {
        "esv": "0x62",
        "operations": [
            {
                "epc": "0xF0"
            }
        ]
    }
}
```

■ レスポンス

```
{
    "response": {
        "esv": "0x72",
        "operations": [
            {
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
                ]
            }
        ]
    }
}
```

SetGet の例

■ リクエスト

```
POST /elapi/v1/devices/<deviceId>/echoCommands HTTP/1.1
{
    "request": {
        "esv": "0x6E",
        "operations": [
            {
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
            ]
        }
    ]
  }
}
```

■ レスポンス

```
{
    "response": {
        "esv": "0x7E",
        "operations": [
            {
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
                ]
            }
        ]
    }
}
```

表 5-3 コード指定によるリクエスト内容

Property	Type	Required	Description	Example
request	object	Yes	リクエストを指定	—
request.esv	string	Yes	ESV を 16 進数の文字列で指定	"0x62" or "0x6E"
request.operations	array	Yes	複号電文を配列で指定可能	—
request.operations.epc	string	Yes	EPC を 16 進数の文字列で指定	"0xF0"
request.operations.edt	array	Yes(Set のみ)	EDT を 16 進数の文字列で指定。配列指定可能	["0x12","0x34", "0x56"]

表 5-4 コード指定によるレスポンス内容

Property	Type	Required	Description	Example
response	object	Yes	レスポンスを指定	—
response.esv	string	Yes	ESV を 16 進数の文字列で指定	"0x72" or "0x7E"
response.operations	array	Yes	複号電文の応答を配列で指定可能	—
response.operations.epc	string	Yes	EPC を 16 進数の文字列で指定	"0xF0"
response.operations.edt	array	Yes	EDT を 16 進数の文字列で指定。配列指定可能	["0x12","0x34","0x56"]

複合電文の記述例についても下記に示す。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

複合電文の例

■ リクエスト

```
POST /elapi/v1/devices/<deviceId>/echoCommands HTTP/1.1
{
    "request": {
        "esv": "0x62",
        "operations": [
            {
                "epc": "0xF0",
            },
            {
                "epc": "0xF2"
            }
        ]
    }
}
```

■ レスポンス

```
{
    "response": {
        "esv": "0x72",
        "operations": [
            {
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
                ],
            },
            {
                "epc": "0xF2",
                "edt": [
                    "0x42"
                ]
            }
        ]
    }
}
```

５．６ エラー処理

HTTP では、レスポンス時のステータスコードを表 ５-５のように規定している。これらを考慮し、

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

サーバは適切にエラーコードなどを返却すべきである。

表 5-5 HTTP Status Code

Status Code	名称	意味
100 番台	Informational	処理が継続中
200 番台	Successful	正常終了
300 番台	Redirection	転送の要求
400 番台	Client Error	クライアント原因のエラー
500 番台	Server Error	サーバ（宅内コントローラや機器との通信含む）原因のエラー

より詳細なエラー情報を返却できる場合には、表 5-6 を参考に適切なステータスコードを返却しても良い（オプション）。

表 5-6 HTTP Status Code の活用事例

Status Code	名称	意味（例）
200	OK	PUT、GET 時の成功
201	Created	POST 時の成功 #Sub リソース作成時
204	No Content	PUT、DELETE 時の成功 #更新、Sub リソース削除時
301	Moved Permanently	リソースは恒久的に移動
304	Not Modified	リソースは未更新 #条件つき GET でキャッシュ使用した場合
400	Bad Request	リクエストが不正 データ形式間違い
401	Unauthorized	認証が必要
404	Not Found	リソース（該当するパスやプロパティ）が存在しない。
405	Method Not Allowed	リソースに指定されたメソッドが許可されない
406	Not Acceptable	Accept ヘッダとマッチしない
409	Conflict	リソースが矛盾（ID が衝突など）PUT 時#更新リソースがロック中
415	Unsupported Media Type	データ形式は正しいがサーバが対応しない
500	Internal Server Error	サーバサイドでエラーが発生
503	Service Unavailable	サーバが一時的に停止

本ガイドラインでは、上記ステータスコードが 4XX 系、5XX 系の場合は更に、必要に応じて ECHONET Lite Web API の呼び出しや、サービスに起因するエラー応答もレスポンスボディ（JSON 形式）にて返却可能とする（オプション）。

下記に示す通り、エラータイプとメッセージ（任意の文字列）を組として応答する。

```
{
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
    "type":<ErrorType>,
    "message":<ErrorMessage>
}
```

表 5-7 サービスレベルのエラー応答

Property	Type	Required	Description	Example
type	string	Yes	Error の Type を示す。サーバが Error と判断する場合（rangeError/referenceError/typeError/timeoutError）と機器がエラーと判断する場合（deviceError）がある。	"rangeError"
message	string	Yes	ERROR の詳細を記述する任意の String data	

type には、下記種類を想定する。

表 5-8 エラータイプの種類

ErrorType	Description	Example
クライアント（クライアント）に起因		
rangeError	設定する値が仕様の範囲外の場合	number, integer：値が min と max の間にない場合 enum: 値が存在しない場合
referenceError	対象とする deviceId や propertyResourceName が存在しない場合	
typeError	設定する値の型が Device Description に記述された型と一致しない場合	
サーバ（コントローラや機器の通信含む）に起因		
timeoutError	機器から一定時間内に返答がない場合	通信系エラー
deviceError	機器から受信したデータが error に対応する値の場合。機器から SNA を受信した場合	SetGet_SNA, Set_SNA を受信した場合

例

```
{
    "type": "rangeError",
    "message": "data is out of range"
}
{
    "type": "rangeError",
    "message": "no value matches to the data"
```


© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

```
}

{
    "type": "referenceError",
    "message": "deviceName is wrong"
}
{
    "type": "referenceError",
    "message": "propertyResourceName is wrong"
}
{
    "type": "typeError",
    "message": "data should be boolean"
}

{
    "type": "timeoutError",
    "message": "timeout !"
}

{
    "type": "deviceError",
    "message": "undefined"
}
{
    "type": "deviceError",
    "message": "GET_SNA"
}
```

５．７ 機器情報

Appendix（別紙）にて記載する。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

第６章 おわりに

本書では、宅内の ECHONET Lite モデルをクラウドなどのサーバ上にマッピングし、宅内の ECHONET Lite 対応機器などを Web API 化したサービスを介して外部クライアントから活用可能とするシステムの構築に際し、Web API 設計時に考慮すべき各種指針について示した。

本書は第１版ガイドラインであり、宅内の機器を操作するユースケースを中心に策定したが、今後策定予定の第２版ガイドラインでは、今回スコープ外としたユースケース（機器の仮想化対応、EMS 情報・稼働履歴情報の収集・加工・検索など）にフォーカスをあて、提供可能なサービスの幅を広げ、より高機能なサービスを提供できるよう、ガイドライン策定を進めていく。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED

第7章 謝辞

本書を作成するにあたり、神奈川工科大学の藤田裕之氏、大和田茂氏には、WG へオブザーバ参加いただき、検討・執筆に際し多大なるご協力を賜りました。深謝いたします。また、議論に参加いただいた関係各位に感謝の気持ちと御礼を申し上げます。

© 2018 ECHONET CONSORTIUM ALL RIGHTS RESERVED