Diagnosis System Developer Guide

2016/12/14

Hinodeya Institute for Ecolife co. ltd.

+81-75-708-8152

hinodeya@hinodeya-ecolife.com

　本システムは、家庭や中小事業所等での二酸化炭素排出削減対策を行うための対策提案ツールを作成するものであり、非営利・非政府の開発・利用について無償で提供される。

# Data Save format in localstrage

localStorage　sindanXXYY

XX language Code

YY Country Code

D6.doc.loadDataSet( decodeURIComponent(escape(atob(evval.rdata))) );

# files

[Diagnosis Logic]

D6/Diagnosis.js

D6/base/

[PHP degign]

index.php

[logic]

D6/logic/

[language set]

D6/logic\*

[language set]

/language/ .php .js .html

Web worker

[facade]

D6/d6facade.js

[view & controler]

View/Main.js

Or direct call

[view degign set]

/view/ .html .css .js

[data format]

D6/base/disp\*.js

# 概要

　日本では、家庭での有効な温暖化対策を提案するための診断ソフトとして、2008年に「うちエコ診断ソフト」が作成された。当初は兵庫県エリアでのモデル的な実施であったが、2011年以降は、国レベルで取り上げられ、診断士制度とあわせて全国で展開されるようになり、年間数千件の診断が行われ、家庭において10%から20%程度CO2削減ができる対策が実行に移されている。

　家庭ごとにエネルギーの使い方は異なり、有効な対策やその効果も異なってくる。その家庭にとって、何が有効な対策なのかを評価するにあたり、エネルギー機器等の状況についてアンケートを行い、家庭ごとに130項目程度の対策を評価して有効な対策を選び出している。

　具体的なCO2削減効果、および光熱費削減効果を示し、組み合わせによりどこまで削減ができるのかを各家庭で選択できるようにしている。

提案される対策例

　・太陽光発電装置の屋根への設置

　・窓の断熱（二重窓、複層ガラス、断熱シートの設置）

　・節水型シャワーヘッドへの取替

　・省エネ家電製品の導入（エアコン、冷蔵庫、テレビ、照明）

　・省エネ型給湯器の導入（ヒートポンプ式、潜熱回収型、コジェネ）

　・冷暖房設定温度の見直し

　・シャワー利用時間の縮減

# Diagnosisシステムについて

　うちエコ診断で使われた診断方法について、その計算ロジックや設定値について見直し、自由な設計ができるように新たな診断ソフトプラットホームとして整備をしたものである。ロジックの基本構造を整理し、省エネ対策計算処理としてよく利用される計算ロジックを整理してバックエンドで自動的に実行できるようにし、簡易に対策を追加削除したり、言語を切り替えたり、パラメータ等を変更したりしやすくする構造とした。また、計算ロジックと表示ロジックを完全に切り分けて、多様な診断形態に対応できるようにした。

　開発ベースとしては、htmlとjavascriptを利用しており、PCだけでなくスマホ等でも実用的な速度で診断ができるほか、インターネットに接続しなくても診断ができるよう設計することも可能となっている。

　規定の方法で、ロジックを記述することにより、多様な地域や言語にも対応できるようにしている。また分野などを見直すことにより、中小事業所向けの診断としても設計可能であり、開発もされている。

# 権利

　ソースコードの著作権は、当面有限会社ひのでやエコライフ研究所が保有する。弊社で開発をすすめた上で、遅くとも2026年までにはフリーソフト化をする予定としている。

　本システムは、非営利・非政府の活動においては、自由に改変し利用することができる。営利事業に用いる場合、また政府組織において改変・利用する場合においては、別途弊社との契約を必要とする。

# システムの構成

## 開発に必要な環境

　・WEBサーバー

　・PHP

　・Microsoft EXCEL

## 診断利用に必要な環境

　・WEBサーバー

※htmlとして出力した場合、ファイルを全て設置することで不要となる場合もある

　・WEBブラウザ（IE9以降、その他モダンブラウザ）

※開発によってはアプリ化も可能

## ファイル構成

index.php 基本設定、開発・テスト用のシステムの切り替え（開発時）

design viewおよびcontrollerに関するフォルダ

css cssデザインフォルダ

templates テンプレートフォルダ

js view controllerに関するjavascript記述フォルダ

main.js controller

tools javascriptに関するviewに関する外部ツールフォルダ(jQuery等)

d6 model計算ロジック

d6facade.js d6システムのファサード(web workerの窓口)

diagnosis.js modelのメイン設定。D6オブジェクトの集約。

base コアロジック

areaset 地域設定フォルダ

disp 表示用結果データ作成ルーチンフォルダ

logic 計算ロジック、地域設定の変更

senarioset.js 対策・入力の総合設定ファイル

　design/javascriptフォルダ内、d6フォルダ内のjavascriptファイルについては、まとめて圧縮して提供される場合がある。

### 使用言語や国の変更

　使用言語や国の変更するためには、以下のファイルを変更することで対応できる。

（ロジック）

d6/logic/senarioset.js 後述のEXCELファイルから設定を転記する

area/\*.js 地域区分や気温、日射量などを設定する

base/unit.js エネルギー種や、単位などの設定をする

（表示）

design/\*.js 画面・グラフ表示ロジック

design/design.html 画面基本構成

### 対策項目や質問・選択肢等の変更

　対策項目や質問・選択肢等を変更するためには、以下のファイルを変更することで対応できる。

（ロジック）

d6/logic/senarioset.js 後述のEXCELファイルから設定を転記する

d6/logic/\*.js 分野ごとに対策効果の計算式等を記述する

### 画面表示の変更

　画面表示を変更するためには、以下のファイルを変更することで対応できる。

（表示）

design/\*.js 画面・グラフ表示ロジック

design/design.html 画面基本構成

## 定義開発用EXCELシート

　senaroset.jsファイルを生成するためのEXCELシート。対策名とアンケート入力内容について記載を行い、ロジックについては、d6/logic/内の各ファイルで記述を行う。

# 分野ファイルの設定

　分野は、d6/logic/に一つずつファイルとして作成し、ファイルごとに規定の方法でクラスを定義する。

## senario.jsの構造

　診断全体のシナリオを定義し、システムの骨格を記述している。詳細の記述については、後述のEXCELファイルで生成して貼り付ける。

D6.senario =

{

//シナリオ記述配列

defCons : [],

defMeasures : [],

defEquipment : [],

defEquipmentSize : [],

defInput : [],

defSelectValue : [],

defSelectData : [],

defQuesOrder : [], //質問順の配列（展開前）

//消費量計算クラスの実装

getLogicList : function()

{

var logicList = new Array();

//分野設定を追加した場合にはここに記述する

//　順番が影響している場合がある

logicList["consTotal"] = D6.consTotal;

logicList["consEnergy"] = D6.consEnergy;

logicList["consSeason"] = D6.consSeason;

logicList["consHWsum"] = D6.consHWsum;

・

（中略）

・

logicList["consCKsum"] = D6.consCKsum;

　ファイルで定義された消費クラスを、getLogicList関数で読み込む。

return logicList;

},

//クラス内容の定義

setDefs : function()

{

var defMeasures = [];

var defInput = [];

var defSelectValue = [];

var defSelectData = [];

var defQuesOrder = [];

var defEquipment = [];

var defEquipmentSize = [];

・

（中略）

・

defMeasures['mTOsolar'] = { mid:"1", name:"mTOsolar", title:"太陽光発電を設置する", easyness:"0.5", refCons:"consTotal", titleShort:"太陽光発電", joyfull:"いくら省エネしても、エネルギーを使わずに生活できるわけはなく、省エネを進めるほど苦しくなるものです。そんな中、自分で「発電する」ことができれば、いともかんたんにゼロが達成できます。いままで電気代を払っていたものが、電気を生産して売れるというのも楽しいですし、それでいて環境にいいことができているわけです。貯金をするよりも利子がつく計算となり、太陽がふりそそぐ屋根をもっていたら、これを逃す手はありません。", level:"", figNum:"25", lifeTime:"20", price:"400000", roanShow:"", standardType:"", hojoGov:"", advice:"発電で余った電気は、電力会社に高く買い取ってもらうことができます。2016年度は1kWhあたり31円（東京電力、中部電力、関西電力の場合）、もしくは33円（それ以外の電力会社では高いですが、太陽光が余剰となったときに買い取りを停止する装置の設置が必要になります）。パネルを設置するだけで発電がされ、モーターなど稼働する部分がないために寿命が長く、維持管理も比較的少なくて済みます。交流に変換をする「コンディショナー」と呼ばれる装置は10年程度ごとに交換が必要になります。<br>　また、太陽光発電装置を導入すると、電気を売る様子が表示される装置が設置されます。電気をどれだけ発電できたのか、家庭でどれだけ消費したのかが表示され、機種によっては時間帯別に表示されるものもあります。販売できた金額も表示され、より多く販売するために自然と電気の使用量が減る効果も出てきます。", lifestyle:"", season:"wss"};

・

（中略）

・

　setDefs関数内に、個別の対策内容や、入力質問・選択肢等を設定します。これらの内容は、EXCELシートで管理して、一括してコピーすることができる。

defInput['i010'] = { cons:'consTotal', title:'対策として重視する視点', unit:'', text:'どんな対策を優先的に表示しますか', inputType:'sel010', right:'', postfix:'', nodata:'', varType:'Number', min:'', max:'', defaultValue:'-1'}; defSelectValue['sel010']= [ '選んで下さい', 'CO2削減優先', '光熱費削減優先', '取り組みやすさ考慮', '取り組みやすさ優先', '', '', '', '', '', '', '', '', '', '', '' ]; defSelectData['sel010']= [ '-1', '1', '2', '3', '4', '', '', '', '', '', '', '', '', '', '', '' ];

・

（中略）

・

//実装化

this.defMeasures = defMeasures;

this.defInput = defInput;

this.defSelectValue = defSelectValue;

this.defSelectData = defSelectData;

this.defEquipment = defEquipment;

this.defEquipmentSize = defEquipmentSize;

}

};

　対策と同様に、入力値についても一括して定義している。入力については

defInput：質問の概要の定義

defSelectValue: 選択肢キャプション

defSelectData: 選択時の返り値

　として、質問事にセットで定義している。

　また、EXCELシートでは生成できない、選択肢の多い項目（都道府県名）についても、直接記述をすることによって対応している。

//平均算出時に条件として考慮する項目

this.defCalcAverage = [ "i001", "i005", "i021"];

　defCalcAverageは、平均値を算出するときに使われる変数を列挙している。この変数が変更になった場合には、平均値算出を自動的に行う。

//対策の提案評価方法

this.measuresSortChange = "i010"; //変数

this.measuresSortTarget = [

"co2ChangeOriginal",

"co2ChangeOriginal",

"costTotalChangeOriginal",

"co2ChangeW1Original",

"co2ChangeW2Original"

];

　i010に関しては、対策のソートをする計算方法として予約してあり、その計算ロジックもそれぞれ用意してある。

//追加入力画面設定

// {}配列で複数設定可能

//

this.defEasyQues = [

{ title:"簡易入力",

cname:"easy01",

ques: [

'i010',

'i021',

・

（中略）

・

　defEasyQuesは、分野と別に、簡易に診断できる入力ページを設ける場合に設定する。標準では入力タブの最初に表示される。

title 画面に表示されるタイトル

cname プログラム内でタブ画面指定に用いられるコード

ques 表示する入力コードを列挙する配列

## cons\*.jsの構造

　消費量を計算する単位で、ファイルを分け、ファイルごとにオブジェクトを一つ作成する。ファイル名（.jsを除く部分）と、オブジェクトは一致させる。

　ファイル名は、consXX\*.js　とし、XXは2文字の大文字とする。消費分野で構わないが、冷暖房については、冷房と暖房を分けたり、合わせてACとして部屋の消費を示す文字を入れたりするなど、分かる範囲で自由に設定して構わない。全体を表すconsTOTALだけは、2文字ではない。

　以下、consACcool.jsファイルとして例を示す。

### オブジェクトのコピー

D6.consACcool = D6.object( D6.consCOsum );

DC = D6.consACcool;

　ファイル名と同じconsACcoolオブジェクトを、D6オブジェクトに生成する。D6.object()は、オブジェクトをコピーするもので、ここでは家の冷房全体の計算を行うD6.consCOsumオブジェクトを使いまわすためにベースとしている。通常は、D6.ConsBaseを基本として継承している。さらにD6.ConsBaseは、D6.Energyを継承しており、電気やガス、CO2量、価格などの値を保持することができる。

　このオブジェクトは何度も使われるため、DCという名前で以降はアクセスする。

### 初期設定 init

//初期設定値

DC.init = function(){

//構造設定

this.consName = "consACcool"; //分野コード

this.consCode = ""; //うちわけ表示のときのアクセス変数

this.title = "部屋冷房"; //分野名として表示される名前

this.orgCopyNum = 1; //初期複製数（consCodeがない場合にコピー作成）

this.addable = "冷暖房する部屋"; //追加可能

this.sumConsName = "consCOsum"; //集約先の分野コード

this.sumCons2Name = "consAC"; //関連の分野コード

this.groupID = "2"; //うちわけ番号

this.color = "0x0000ff"; //表示の色

this.countCall = "部屋目"; //呼び方

this.inputGuide = "部屋ごとの冷房の使い方について"; //入力欄でのガイド

//対策の関連定義

this.measureName = ["mCOtemplature"];

};

DC.init();

　DC.init()関数では、オブジェクトの基本設定を行う。

this.consName = "consACcool";

消費クラスの名前を記載する。

this.consCode = "";

　消費クラスにアクセスするための2文字の大文字を設定するが、消費内訳の表示で用いられるようなconsTOTALに直結するオブジェクトの場合のみ記載する。関連しない場合には空白とする。

this.title = "部屋冷房";

　分野名のタイトル。

this.orgCopyNum = 1;

　初期設定時にこのオブジェクトの消費実態を何個作成するのかを記載する。部屋であったり、機器台数であったり、数が増やせるものについて設定を行う。通常は1とし、後から増やすこともできる。

this.addable = "冷暖房する部屋";

　部屋や機器台数など、このオブジェクトの消費実体を増やすにあたって、何を増やすのかを文字で示す。

this.sumConsName = "consCOsum";

　集約先の消費オブジェクト名を記述する。ここでは各部屋の冷房のオブジェクトであるので、集約先としては「冷房」オブジェクトが用いられる。

this.sumCons2Name = "consAC";

　集約先の消費オブジェクト名として、メインではないが集約されるオブジェクトを記述する。各部屋の冷房消費は、各部屋の暖房消費と合わせて、「部屋の冷暖房」オブジェクトの一部として扱う。

this.groupID = "2";

　消費分野の番号で0から9までの数値。

this.color = "0x0000ff";

　内訳グラフで色分けをする場合の色。ただし、内訳表示に関わるオブジェクトでないと意味はない。

this.countCall = "部屋目";

　オブジェクトの消費実体を増やす場合の、数の呼び方。

this.inputGuide = "部屋ごとの冷房の使い方について";

　画面表示されたときに、表示するガイド。

this.measureName　対策として計算を行うオブジェクトの名前をここに列挙する。

### 消費量計算 calc

//冷房消費量計算

DC.calc = function() {

this.clear(); //結果の消去

//入力値の読み込み

//全体なので、部屋毎の設定は読み込まない

this.houseSize = 1;

this.coolArea = this.input( "i212" + this.subID, 12 ); //部屋面積

this.coolTime = this.input( "i271" + this.subID, this.sumCons.coolTime ); //冷房時間

this.coolTemp = this.input( "i273" + this.subID, this.sumCons.coolTemp ); //冷房温度

if ( this.subID != 0 ){

this.mesTitlePrefix = this.subID + "部屋目";

}

var coolKcal = this.calcCoolLoad();

//年平均値への換算（季節分割をしている場合にはここでするべきではない）

coolKcal \*= D6.area.seasonMonth.summer / 12;

this.electricity = coolKcal / this.apf / D6.Unit.calorie.electricity;

//月の消費電力量　kWh/月

};

　calc関数で消費量の計算と、対策計算における基本数値の算出を行う。

　まず、clear()で値のクリアを行う。

　input()　関数で入力ダイアログ（HTML）の値を読み込んで設定することができる。保存する変数名については、重複しない限り特に制約はないが、this.\*として変数を設定するとこのオブジェクトを通じて他からもアクセスができる。対策計算でも使わない値については、input関数内でのみ使用できる変数として、varで変数宣言をしても構わない。

　input()関数のパラメータの1つ目は設定された入力変数名だが、this.orgCopyNumに1が設定してあり、部屋数を増やせるオブジェクトであるため、複製番号としてthis.subIDを追記した形で保存されており、それを呼び出している。パラメータの2つめを無記入(-1)だった場合の標準値。

　this.subIDについては、this.orgCopyNumを参考に、diagnosisオブジェクトで初期設定されるときに、複製番号として自動的に設定される。

　mesTitlePrefixは、消費量のタイトルを表示するにあたって、属性をタイトルの前に表示するその内容。

　消費量クラスとして、各エネルギー消費量についてはここで計算をしてしめす。this.electricityは電気の消費量。同様に、this.gas、this.keroseneなど、消費があるエネルギーについては値を設定しておく。clear()されているので設定しない場合には、0が初期設定された状態で使われる。

### 消費量計算の後処理 calc2nd

//calc計算が全てのクラスで終了した後に実行

DC.calc2nd = function( ) {

//残渣の場合の処理

if ( this.subID == 0 ){

this.electricity = this.sumCons.electricity;

var cons = D6.consListByName[this.consName];

for( var i=1 ; i< cons.length ; i++ ){

this.electricity -= cons[i].electricity;

}

}

};

　calc2ndは、一度すべての消費量計算がされた後、他のオブジェクトとの関係などを考慮して設定しなおす場合に記述する。

　ここでは、subIDが0の場合には、合計消費量から、他の複製数分の消費量の合計を引いた残差分として設定している。通常は複製が可能な消費オブジェクトで、subIDが0のものは、残差として扱う。

　this.sumConsは、sumConsNameで設定した集約関数名の消費実体であり、その数値を呼び出すことができる。

### 対策計算 calcMeasure

//対策計算

DC.calcMeasure = function() {

if ( this.subID > 0 || this.electriity == this.sumCons.electricity ) return;

//mCOtemplature

if ( this.coolTemp < 28 && this.coolTemp > 20 ){

this.measures["mCOtemplature"].calcReduceRate( ( 28 - this.coolTemp ) / 10 );

} else {

this.measures["mCOtemplature"].calcReduceRate( 0 );

}

};

　calcMeasureで対策の計算を行う。この分野に関わる複数の対策をここでまとめて計算することができる。各対策の計算については、this.measuresのオブジェクトで、対策オブジェクトの名前をキーに指定する。このとき、initでオブジェクト名の配列を設定しておかないと対策オブジェクトが生成されず、この関数内でアクセスができずエラーとなる。

　calc等で算出した値や、initで設定した値はここで呼び出して、計算に使用することができる。

　this.measures["mCOtemplature"]　は対策オブジェクトを指しており、主に以下の関数や変数で設定ができる。対策オブジェクトは D6.MeasureBaseを継承している。またさらにD6.MeasureBaseはD6.Energyを継承しており、電気やガスなどの消費量も直接保持することができる。

calcReduceRate(rate) 現状の消費オブジェクトに対して、rateの比率で割引を行った値とする

0の場合には、現在の消費量と同じになるため、削減されないことを意味する。

electriciy この対策を導入した後の、この消費オブジェクトの消費量がどのように変化するのかを記述するにあたって、電気消費量を直接指定する。同様にガス、灯油なども設定できる。

cons\* 対策による消費オブジェクトへの変更を、他の消費オブジェクトにどのように割り振るのかを設定するために、消費オブジェクト名を指定して設定ができる。この変数を、D6.Energyオブジェクトとして生成し、電気・ガスの消費量などを設定する。対策が選択された場合には、それぞれの消費オブジェクトの消費量値が、ここで設定した値に置き換えられる。

priceNew 導入価格を設定する。

　なお、値の設定においては、他の対策の選択状況によっても効果が異なってくる場合がある。この消費オブジェクトから　isSelected(“mXX\*”)で選択されているかどうかを判定できる。

### 計算実行のタイミング

　initについては、このオブジェクトの生成時点で実行がされる。

　calc,calc2nd,calcMeasureについては、diagnosis.jsの関数から、指定されたときに消費オブジェクトをまとめて実行される。

# EXCEL設定

　入力項目（質問）、対策について、一覧表の形式で整理を行う。この整理されたシートでコードが生成できるため、senarioset.jsファイルに簡単に転記することができる。

## シート構成

記述規則 0から9までの分野を定義する、共通で使用する対策コード（英字大文字2文字）

対策一覧 対策を1行1項目で記載し、jsに転記するコードを生成する

入力設定 入力質問を1行1項目で記載し、jsに転記するコードを生成する

入力番号整理 確認のため入力番号について一覧で表示する

## 入力設定

　1行に1項目を入力する。A列からI列でsenarioset.js出力用のコードが生成される。

### O列~AA列　質問の設定



画面・分野：関連する消費クラス名

入力変数名：iと数値3桁で定義する。100位については、消費分野ごとに0から9まで分類がされている。ただし強制力をもつものではない。

設問：質問一覧に記載される質問のタイトル。

単位：入力をするときの単位を記述する。なくてもよい。

質問内容：入力時にガイドとして表示される内容。

入力方法：選択肢から選ぶ場合には、その配列コード。

テキスト右づめ：数値など右づめで表示する場合に 1をセットする。なくてもよい。

入力処理：入力した値を、確定時に変換する場合の設定。

-1のとき表示：初期値が-1等である場合、それを記入欄に表示するかどうかを決める。

保存形式：Number、String、Booleanなど変数形式を記述する

範囲最小：数値の範囲としてこれ以下の場合に、処理を行う。

範囲最大：数値の範囲としてこれ以上の場合に、処理を行う。

初期値：未記入の場合の初期値を記載する

消費分野コード

0 太陽光・契約・家庭全般

1 給湯・水道

2 冷暖房

3 調理

4 洗濯・衣類乾燥

5 照明

6 テレビ

7 冷蔵庫

8 自家用車

9 待機電力・その他

### AD列~AT列　選択肢キャプションの設定



　選択肢IDは、質問定義で記載された「入力方法」の値を記載し、通常は”sel”の後に入力変数名で使った数値3桁を用いる。「入力方法」で対応がされていれば、設問と選択肢の数値3桁が一致していなくても動作するが、一致しているほうが管理しやすい。

　選択肢の0は、初期状態で表示されるものとし、入力を促す表記があることが望ましい。

　1から15まで選択肢の表記を入力できるようになっている。順番に記載し、選択項目が終わる場合にはそれ以降の記載はしない。15項目以上の選択肢がある場合には、ここでは処理ができないため、下記のように直接senarioset.jsに記述する。

defSelectValue['sel021'] = [ "選んで下さい", "北海道", "青森", "岩手", "宮城", "秋田", "山形", "福島", "茨城", "栃木", "群馬", "埼玉", "千葉", "東京", "神奈川", "新潟", "富山", "石川", "福井", "山梨", "長野", "岐阜", "静岡", "愛知", "三重", "滋賀", "京都", "大阪", "兵庫", "奈良", "和歌山", "鳥取", "島根", "岡山", "広島", "山口", "徳島", "香川", "愛媛", "高知", "福岡", "佐賀", "長崎", "熊本", "大分", "宮崎", "鹿児島", "沖縄" ];

defSelectData['sel021']= [ '-1', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47' ];

### AU列~BJ列　選択変数の設定



　選択肢キャプションの右に続けて記載する。選択肢が選択された場合に、返す値を定義する。

　初期状態の0については、通常は-1を設定する。ただし、回答でマイナス数値を取りうる場合には、それ以外の値を設定することもできる。

## 対策設定

　対策について1行1項目を設定する。X列にソースコードが出力される。

### B列~K列



対策手段ID：重複がない3桁の数値でコードを示す。100の位は、消費分野に相当する0から9を割り当てる。ただし強制力はない。

対策分野：消費分野に相当する0から9の数値。

対策分野名：自動的に「記述規則」シートから呼び出される。

対策コード：対策ごとに割り振る文字。”m”と英大文字2文字の文やコードに続き小文字で対策名を記載する。

タイトル：対策のタイトルを記載する。

取り組みやすさ：取り組みやすいものを3、取り組みにくいものを1とする。対策表示の設定により、CO2排出量に掛け合わせる。特に取り組みにくいもの、取り組みやすいものは、この数値の比を参考に、0.5、5などの値を設定しても構わない。

関連消費クラス：ここでは設定せず、消費ファイルで関連を設定する。

短いタイトル：対策内容がわかる範囲で最も短いタイトルを記述する

取り組みに関する楽しい話：一般的ではないが、少しくだけた話を記載する。

### L列~T列



診断レベル：0または無記入であれば、常に結果を表示する。簡素化診断をするレベルに応じて、数値を大きくする。

図番号：画像として表示をするイラストの番号を定義する。この番号をもとに画像を用意して表示をできるようにする。

寿命：通常の利用における使用年数を記載する。白熱電球やLED等については使用時間で寿命が決まっているものがあり、これは1000hなど最後にhをつけることで時間単位であることを記載する。

価格：初期投資の価格を記載する。太陽光発電装置などについては、設置kwあたりの単価とし、消費クラスにおいて、設置サイズに応じて価格を設定する。また、機種ごとにサイズ等によって価格が異なる場合には、消費クラスで設定をし、ここでは初期値とする。

ローン：ローンを組んで購入をする可能性のあるものを記載する。ローン評価をする画面を生成するときには、この値を参考にする。

比較普及機種名：比較対象とする普及型機種の名前を記載する。普及型機種と比較する場合には、消費クラスで個別に設定をする。

国補助：国からの設置補助が出る場合には、その内容をここに記載する。

アドバイス：対策の概要と取り組み方等についての記載をする。

ライフスタイル項目：投資を必要とするものでなく、生活の工夫によって取り組める項目の場合に、ここに1を記入する。

提案シーズン：対策の提案をするシーズンを制限する。wssの3文字で左から冬、春秋、夏を意味し、その季節に提案をしないのであれば、文字の代わりに0を記載する。例えば、00sであれば夏のみに提案を表示する。

# 地域設定 Area\*

　地域により電力契約や、気候などが異なっている場合には、それぞれの設定を記述できるようにしている。都道府県、州、地方など別に各種の値を設定できる。電力料金体系については、地域にかぎらず複数の設定がある場合にはここで定義をして、選択できるようにする。

# 単位設定 unit.js

　エネルギー種とその名称、単価、CO2係数、エネルギー係数などを設定することができる。電気については、電力会社や契約によって異なる料金体系であることが多く、係数をareaで設定をおこなう。

D6.Unit = {

co2 : {

electricity:0.55,

nightelectricity:0.55,

sellelectricity:0.55,

nagas:2.23,

lpgas:5.98,

kerosene:2.49,

gasoline:2.32,

lightoil:2.62,

heavyoil:3,

coal:0,

biomass:0,

hotwater:0,

waste:0,

water:0,

gas:2.23,

car:2.32

},

　CO2排出係数を記載する。入力する単位に合わせた値を設定する。

price : {

electricity:22,

nightelectricity:8,

　単価を設定する。

priceBase : {

electricity:0,

nightelectricity:2100,

　基本料金を設定する。基本料金＋単価×消費量を請求額とする。ただし、ここでは1ヶ月ごとの支払いとして計算している。

name : {

electricity:"電気",

nightelectricity:"夜間電気",

　表記する名前を設定する。

unitChar : {

electricity:"kWh",

nightelectricity:"kWh",

　記入する単位を設定する。CO2係数や、一次エネルギー係数はこの消費単位に対する値とする。

//kcal/各単位（二次エネルギー）

calorie : {

electricity:860,

nightelectricity:860,

　二次エネルギー係数（kcal）を記述する。

//　一次エネルギーMJ/各単位

jules : {

electricity:9.6,

nightelectricity:9.6,

　一次エネルギー係数（MJ）を記述する。

# 表示クラスからの呼び出し

　main.jsから　calcStart(params)で呼び出し、d6facade.jsでworkerとして、D6の各関数を呼び出され、値の計算や、表示用結果の取得がされる。

　この値を用いて、グラフや表、入力ページの表示がされる。

## 計算ロジックに関する呼び出し

設定 D6.setSenario() 定義からのロジックの生成

D6.addSet(consName) 分野内の区分（機器・部屋）の追加

consName:分野名コード

計算 D6.calcAverage() 平均値の計算

D6.calcMeasures(cid) 消費量・対策の計算

cid:分野ID,-1なら全体を計算

入力 D6.inSet(id,val) 入力値の設定

id:入力番号（入力変数ID、もしくは入力変数ID＋subID）

val:入力値

D6.measureAdd(mesId) 対策の選択

D6.measureDelete(mesid) 　対策の選択解除

mesid:対策ID

IO D6.doc.serialize() 入力データのシリアライズ

D6.doc.loadDataSet(data) 　保存値の計算式への設定

data:保存値

## 計算結果・画面出力情報作成

入力欄 D6.disp.getInputPage(consName,subName)　分野ごとの入力ページの生成

consName:分野名

subName:分野名-サブ分野名

返り値 group[cname] タブ名、cnameはconsShowに列挙されている消費名

groupAddable[cname] 加算可能なタブ名

consName 追加グループ名

caption 表示

subgroup[cname] サブグループ名

subguide[cname][subgroup] サブページのガイド

combos[cname][subgroup] 質問ごとのコンボボックスhtml

(cnameはconsNameと一致している場合のみ

consName 分野名

subName 分野名-サブ分野名

結果 D6.disp.showResultTable(consName) 結果一覧の表示（デマンド以外）

consName:分野名コード

返り値 graphItemize 内訳グラフデータD6.disp.showItemizeGraph(consCode, sort )

graphMonthly 月変化グラフ

data[n] グラフの一要素

item エネルギー種類

cost 光熱費

month 月

yaxis Y軸の単位

average 平均値

you あなたCO2

after 対策後CO2

av 平均CO2

youc あなた金額

afterc 対策後金額

avc 平均金額

rank100 順位

afterrank100 対策後順位

samehome 同様の家庭に関する情報

sameoffice 同様の事業所に関する情報

consCode 消費コード

cons[i] 分野消費量：消費が多い分野順

title 分野名

consName 分野コード

subID

sumConsName 集約側分野名

sumCons2Name 集約側分野名

countCall

co2 CO2

co2Total 全体でのCO2

electricity

nightelectricity

gas

water

kerosene

car

measure 対策一覧

mesID 対策ID

title 対策名

selected 選択中かどうか

consName 現在表示の消費名

consconsName 関連消費名

conssumConsName　関連消費の集約消費名

conssumCons2Name　関連消費の集約消費名

co2ChangeOriginal　CO2変化

costChangeOriginal　コスト変化

conssubID 分野のサブID

consmesTitlePrefix

relation 他分野に関連付けられているか

inpage D6.disp.getInputPage(consName,subName)

D6.disp.showItemizeGraph(consCode, sort ) 内訳グラフの表示

consCode:分野名コード

sort:表示対象(CO2,Energy,cost)

返り値 data[n] グラフの一要素

value 値(co2,jule,cost)

item 分野

percent 割合

category カテゴリー

ord[n] 分野の表示順

yaxis Y軸の単位

clist[n]

color 色

target 値？

title 分野

averageCaption 平均値の表示

captions カテゴリー名

consTitle タイトル

デマンドD6.disp.getInputDemandSumup() デマンド積み上げ入力

返り値 data[consName][i][j] consName:消費カテゴリー名

i:連番

j:表示位置

title[consName] タイトル

D6.disp.getInputDemandLog() デマンド1時間毎入力

返り値 demandlog[i] i時における数値入力欄

D6.disp.getDemandGraph() デマンドグラフ

log[time] 時間ごとの値

equip 機器

time 時刻

electricity(kW) 消費電力

sumup[i] 機器ごとのグラフ

equip 機器

time 時刻

electricity(kW) 消費電力

clist 色リスト

対策情報D6.disp.getMeasureDetail(id) 対策ごとの詳細情報

id:対策ID

返り値 title

titleShort

measureName

mesID

groupID

advice

joyfull

total (cons.total)

co2Total consTotal.co2

selected

co2

co2Change

co2ChangeOriginal

jules

cost

costChange

costChangeOriginal

costTotalChange

costTotalChangeOriginal

costUnique

priceOrg

priceNew

payBackYear

lifeTime

electricity

gas

car

kerosene

water