Buddy JavaScript API リファレンスマニュアル

2020/08/05
インフォラボ

【概要】

この文書では、Buddy アプリ開発者がスクリーン設計において JavaScript プログラムの開発に用いるアプリケーションプログラムインタフェース（API）について述べます。

【JavaScript API の構成】

Buddy の JavaScript API は、スクリーン設計およびサーバー機能設計に用いられます。

スクリーン設計の JavaScript API は次の構成要素に大別されます。

a. スクリーンオブジェクト
b. スクリーンモジュール
c. モデルオブジェクト
d. apiオブジェクト
e. データベースAPI
f. イベントAPI
g. フィルタAPI
h. 数式API

一方、サーバー機能設計の JavaScript API はサーバー側で実行する処理を記述するために用いられます。

【スクリーンオブジェクト】

アプリを構成する個々の画面を表すオブジェクトのことを「スクリーンオブジェクト」と呼びます。基本的に１つのスクリーンオブジェクトは１つの画面に対応します。

【スクリーンプログラム】

スクリーン設計において Buddy アプリ開発者が作成する JavaScript プログラムのことを「スクリーンプログラム」と呼びます。スクリーンプログラムは以下のテンプレートに基づいて作成します。

```
module.exports = function(api){
        var actions = {
                // ここにアクションを記述します
        };
        var events = {
                onLoad: function(){
                },
                onUnload: function(){
                },
                onResize: function(){
                },
                // ここにイベントハンドラを記述します
                radioname: {
                        // ここにラジオボタン群のイベントハンドラを記述します
                }
        };
        var formulas = {
                // ここに数式ハンドラを記述します
        };
        return {
                "actions": actions,
                "events": events,
                "formulas": formulas
        };
};
```

スクリーンプログラムは、全体が 1 つの関数定義になっています。この関数の形で記述されるオブジェクトを「スクリーンオブジェクト」と呼びます。関数の api 引数には、スクリーンプログラム開発に用いる種々の JavaScript ライブラリへのアクセスを提供する「api オブジェクト」が渡されます。api オブジェクトの詳細については【api オブジェクト】の節を参照してください。

var actions = { ... から始まるブロックにはアクションを記述します。アクションには、スクリーンオブジェクトのプログラム中で繰り返し使うサブルーチンや、スクリーンの外側に公開する関数などがあります。

var events = { ... から始まるブロックにはイベントハンドラを記述します。テンプレートには onLoad、onUnload、onResize の 3 つのイベント（後述）に対するハンドラの雛形があらかじめ用意されています。ボタンやテキストボックスなどのスクリーンモジュールから発生するイベントを受け取るためのイベントハンドラもこのブロックに記述しま

す。イベントハンドラの記述方法については【イベント API】の節を参照してください。events ブロック内の radioname: { ... で始まる部分にはラジオボタン群のイベントハンドラを記述します。詳しくは【ラジオボタン群の操作】の節を参照してください。

var formulas = { ... から始まるブロックには数式ハンドラを記述します。数式ハンドラの記述方法については【数式 API】の節を参照してください。

【メソッド】

本項ではスクリーンオブジェクトのメソッドについて説明します。

【deserialize】

deserialize メソッドはスクリーンを構成するすべてのモジュールの現在の状態を復元します。

```
this.deserialize(data)
```

data 引数には後述の serialize メソッドで得られる文字列を与えます。

【emit】

emit メソッドはイベントを発出します。

```
this.emit(name, data)
```

name 引数にはイベント名を文字列で与えます。data 引数にはイベントハンドラに渡されるデータ（オブジェクト）を与えます。

【getItemDataForDB】

getItemDataForDB メソッドは DB テーブル・DB ビューのカラムに関連付けられたスクリーンモジュールの値を一括して取得します。このメソッドの呼び出し方法には次の 2 通りがあります。

```
this.getItemDataForDB(modelName)
this.getItemDataForDB(cols, modelName)
```

modelName 引数には DB テーブル・DB ビューの名前を与えます。cols 引数にはカラム名の配列を与えます。戻り値は { カラム名 : 値 , …} の形式のオブジェクトです。

【getScreenInfo】

getScreenInfo メソッドはスクリーンに関する情報を返します。

```
this.getScreenInfo()
```

戻り値は以下のプロパティから成るオブジェクトです。

```
PID        … スクリーンコンテナとボックスの入れ子に対応した名前。
ScreenName … スクリーン名。
ScreenDisplayname … スクリーンの表示名。
ScreenType … スクリーンタイプ("PAGE"または"CONTAINER")。
ScreenLayout … スクリーンレイアウトの種別("box"または"grid")。
ScreenMinWidth … スクリーンオプションの「モバイル対応幅」の値。
```

【getParams】

getParams メソッドは URL のパラメータ部分を返します。

```
this.getParams()
```

戻り値は { 変数名 : 値 , ...} の形式のオブジェクトです。

【getQuery】

getQuery メソッドは URL 末尾のクエリ部分を返します。

```
this.getQuery()
```

戻り値は { 変数名 : 値 , ...} の形式のオブジェクトです。

【serialize】

serialize メソッドはスクリーンを構成するすべてのモジュールの現在の状態をひとつの文字列にまとめます（この処理をシリアライズと呼びます)。

```
this.serialize()
```

別のスクリーンに移って戻ってきたときにスクリーンの状態を元に戻すスクリーンプログラムの例を以下に示します。

```
onUnload: function(){
    api.store["serialize"] = this.serialize();
},
onLoad: function(){
    var data = api.store["serialize"];
    if (data) {
        this.deserialize(data);
    }
},
```

onUnload 関数（Upload イベントハンドラ）はスクリーンを閉じるとき（たとえば別のスクリーンへ移るとき）に呼ばれます。この関数の中で serialize メソッドを用いてスクリーンを構成する全モジュールの現在の状態をシリアライズして api.store モジュールのキーバリューストアに保存します。onLoad 関数（Load イベントハンドラ）はスクリーンを開くとき（たとえば別のスクリーンから戻ってきたとき）に呼ばれます。この関数の中で api.store モジュールのキーバリューストアを調べて、もしシリアライズの結果が保存されていたら、deserialize メソッドを用いてスクリーンを構成するモジュールの状態を復元します。

【setItemDataFromDB】

setItemDataFromDB メソッドは DB テーブル･DB ビューのカラムに関連付けられたスク

リーンモジュールの値を一括して設定します。このメソッドの呼び出し方法には次の 2 通りがあります。

```
this.setItemDataFromDB(modelName, data)
this.setItemDataFromDB(cols, data)
```

modelName 引数には DB テーブル・DB ビューの名前を与えます。data 引数には設定する値を { カラム名 : 値 , …} の形式のオブジェクトで与えます。cols 引数には以下のプロパティから成るオブジェクトを要素とする配列を与えます。

```
name     … スクリーンモジュール名
source   … data引数のカラム名
```

【transitionTo】

transitionTo メソッドは現在のスクリーンから別のスクリーンへ移行します。

```
transitionTo(screenName, params, query)
```

screenName 引数にはスクリーン名を文字列で与えます。params 引数には URL の追加パスとしてデータを渡すときに使用するパラメータを与えます。ただし、どのような追加パスを使用するかをあらかじめ設定しておく必要があり、今のところアプリのスクリーンについてこの設定をする手段が用意されていないため使えません。よって params 引数には空のオブジェクト {} を指定してください。query 引数には URL の末尾に「? 変数 = 値 &...」の形式でデータを渡すためのクエリを { 変数名 : 値 , ...} の形式のオブジェクトで与えます。params 引数と query 引数は省略できます。

【イベント】

本項ではスクリーンオブジェクトから発生するイベントについて説明します。

【Iterate】

Iterate イベントは、スクリーンコンテナモジュールの iterate アクションが実行されたときに生じます。イベントハンドラは次のように定義します。

```
onIterate: function(data, index){ ... }
```

data 引数にはスクリーンコンテナモジュールの iterate アクションの引数に与えたデータ配列の要素、index 引数には何番目の反復かを表す値（最初が 0）が渡されます。

【Load】

Load イベントはスクリーンが表示されたときに生じます。イベントハンドラは次のように定義します。

```
onLoad: function(){ ... }
```

【Resize】

Resize イベントはスクリーンが表示されたときと縦横サイズが変更されたときに生じます。イベントハンドラは次のように定義します。

```
onResize: function(){ ... }
```

【Unload】

Unload イベントはスクリーンが閉じられるときに生じます。イベントハンドラは次のように定義します。

```
onUnload: function(){ ... }
```

【プロパティ】

本項ではスクリーンオブジェクトのプロパティについて説明します。

【items】

スクリーンオブジェクトの items プロパティ（this.items）は、そのスクリーンを構成するボタンやテキストボックスなどの要素（スクリーンモジュール）へのアクセスを提供するオブジェクトです。たとえば、BUTTON1 という名前のボタンモジュールは次のように参照できます。

```
this.items.BUTTON1
```

スクリーンモジュールには各種のアクション（後述）が定義されています。たとえばボタンモジュールには値を設定する setValue アクションが設けられています。このアクションは次のように実行することができます。

```
this.items.BUTTON1.setValue("apple");
```

【models】

スクリーンオブジェクトの models プロパティ（this.models）は、アプリで定義されたテーブルおよびビューの設計情報へのアクセスを提供するオブジェクトです。

【radionames】

スクリーンオブジェクトの radionames プロパティ（this.radionames）は、同じ radioname 属性をもつラジオボタン群について、現在選択されているラジオボタンのアイテム名やキャプション（caption）を得るためのメソッドを提供するオブジェクトです。詳しくは【ラジオボタン群の操作】の節を参照してください。

【スクリーンモジュール】

スクリーンモジュールは、スクリーンを構成する部品を表すオブジェクトです。スクリーンモジュールには、大きく４つの用途別に以下のものがあります。

・レイアウト用モジュール

【横配置ボックス】
【縦配置ボックス】
【自由配置ボックス】
【水平区切り線】
【垂直区切り線】

・データ入力用モジュール

【ボタン】
【データインプット】
【テキストボックス】
【チェックボックス】
【ラジオボタン】
【プルダウンリスト】
【コンボボックス】
【トグルスイッチ】
【スライダー】
【日時選択】
【色選択】
【DBレコードセレクタ】
【タブ】
【メニュー】
【ファイル選択】

・データ表示用モジュール

【画像】
【文字列】
【数式】
【リスト】
【箇条書き】
【テーブル】
【DBテーブル】
【スピン】
【プログレスバー】
【カレンダー】
【タイムテーブル】
【クロス集計】
【地図】
【グラフ】
【キャンバス】

・埋め込み用モジュール

【スクリーンコンテナ】
【IFrame】

※モジュール名をクリックするとそのスクリーンモジュールの節にジャンプします。

【横配置ボックス】

横配置ボックスモジュールは、入れ子になった他のスクリーンモジュール（子モジュール）を横に並べて画面上に配置するレイアウト用のモジュールです。子モジュールは横 1 列分の幅の合計が横配置ボックスモジュールの幅を超えないように折り返して配置されます。

【アクション】

本項では横配置ボックスモジュールのアクションについて説明します。

【setStyle】

横配置ボックスのスタイルを設定します。style 引数にはスタイルオブジェクトを与えます。

```
setStyle(style)
```

たとえば､HORIZONTAL1 というモジュール名の横配置ボックスにスタイルを設定するには次のように記述します。

```
this.items.HORIZONTAL1.setStyle({ width: "300px", height: "400px" });
```

【イベント】

本項では横配置ボックスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

【DoubleClick】

DoubleClick イベントはモジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはモジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

【MouseUp】

MouseUp イベントはモジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

【縦配置ボックス】

縦配置ボックスモジュールは、入れ子になった他のスクリーンモジュール（子モジュール）を縦に並べて画面上に配置するレイアウト用のモジュールです。前項の横配置ボックスモジュールとは異なり、子モジュールの高さの合計が縦配置ボックスモジュールの高さを超えても折り返しはされません。

【アクション】

本項では縦配置ボックスモジュールのアクションについて説明します。

【setStyle】

縦配置ボックスのスタイルを設定します。style 引数にはスタイルオブジェクトを与えます。

```
setStyle(style)
```

たとえば、VERTICAL1 というモジュール名の縦配置ボックスにスタイルを設定するには次のように記述します。

```
this.items.VERTICAL1.setStyle({ width: "300px", height: "400px" });
```

【イベント】

本項では縦配置ボックスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはモジュールがダブルクリックされたときに生じます。イベント

ハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはモジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはモジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【自由配置ボックス】

自由配置ボックスモジュールは、入れ子になった他のスクリーンモジュール(子モジュール）をモジュール枠内の自由な位置に配置できるレイアウト用のモジュールです。

【アクション】

本項では自由配置ボックスモジュールのアクションについて説明します。

【setStyle】

自由配置ボックスのスタイルを設定します。style 引数にはスタイルオブジェクトを与えます。

```
setStyle(style)
```

たとえば、FREE1 というモジュール名の自由配置ボックスにスタイルを設定するには次のように記述します。

```
this.items.FREE1.setStyle({ width: "300px", height: "400px" });
```

【イベント】

本項では自由配置ボックスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはモジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはモジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはモジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【水平区切り線】

水平区切り線モジュールは、画面を区切る横線を引くレイアウト用のモジュールです。

【アクション】

本項では水平区切り線モジュールのアクションについて説明します。

【setStyle】

setStyle アクションは、水平区切り線のスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項では水平区切り線モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントは水平区切り線がクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントは水平区切り線がダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントは水平区切り線の上でマウスボタンが押下されたときに生じま

す。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントは水平区切り線の上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【垂直区切り線】

垂直区切り線モジュール、画面を区切る縦線を引くレイアウト用のモジュールです。

【アクション】

本項では垂直区切り線モジュールのアクションについて説明します。

【setStyle】

setStyle アクションは、垂直区切り線のスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項では垂直区切り線モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントは垂直区切り線がクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントは垂直区切り線がダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントは垂直区切り線の上でマウスボタンが押下されたときに生じま

す。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントは垂直区切り線の上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ボタン】

ボタンモジュールは、画面上にボタンを設けるモジュールです。

【アクション】

本項ではボタンモジュールのアクションについて説明します。

【setDisabled】

setDisabled アクションは、ボタンを押せない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setStyle】

setStyle アクションは、ボタンのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、ボタン内に表示されるラベル文字列を設定します。

```
setValue(value)
```

value 引数にはラベル文字列を与えます。

【イベント】

本項ではボタンモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはボタンから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Click】

Click イベントはモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはモジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはボタンに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはモジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはモジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【データインプット】

データインプットモジュールは、DB テーブルのカラムに関連付けれたデータ入力欄を設けるモジュールです。データの入力方法は､カラムの設定に応じてテキストボックス、チェックボックス、プルダウンメニュー、日付選択などの入力方法の中から自動的に選ばれます。

【アクション】

本項ではデータインプットモジュールのアクションについて説明します。

【getValue】

getValue アクションは、データ入力欄に入っている値を返します。

```
getValue()
```

【setDisabled】

setDisabled アクションは、データ入力欄を使用できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setFocus】

setFocus アクションは、データ入力欄に入力フォーカスを移します。

```
setFocus()
```

【setReadOnly】

setReadOnly アクションは、データ入力欄を読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStyle】

setStyle アクションは、データ入力欄のスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、データ入力欄の値を設定します。

```
setValue(value)
```

value 引数には設定する値を与えます。データ入力欄の値が変わると後述の【Change】イベントが発生します。

【イベント】

本項ではデータインプットモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはデータ入力欄から入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはデータ入力欄の値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントはデータ入力欄がクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはモジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはデータ入力欄に入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyDown】

KeyDown イベントはモジュール上でキーボードのキーが押し下げられたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyPress】

KeyPress イベントはモジュール上でキーボードのキーが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyPress: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyUp】

KeyUp イベントはモジュール上で押下されたキーボードのキーが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはモジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはモジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【テキストボックス】

テキストボックスモジュールは、画面上に文字列の入力欄を設けるモジュールです。

【アクション】

本項ではテキストボックスモジュールのアクションについて説明します。

【getValue】

getValue アクションは、テキストボックスに入っている値を返します。

```
getValue()
```

【setDisabled】

setDisabled アクションは、テキストボックスを使用できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setFocus】

setFocus アクションは、テキストボックスに入力フォーカスを移します。

```
setFocus()
```

【setReadOnly】

setReadOnly アクションは、テキストボックスを読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStyle】

setStyle アクションは、テキストボックスのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、テキストボックスの値を設定します。

```
setValue(value)
```

value 引数には設定する値を与えます。テキストボックスの値が変わると後述の【Change】イベントが発生します。

【イベント】

本項ではテキストボックスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはテキストボックスから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはテキストボックスの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントはテキストボックスがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはテキストボックスがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはテキストボックスに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyDown】

KeyDown イベントはテキストボックス上でキーボードのキーが押し下げられたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyPress】

KeyPress イベントはテキストボックス上でキーボードのキーが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyPress: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyUp】

KeyUp イベントはテキストボックス上で押下されたキーボードのキーが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはテキストボックス上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはテキストボックス上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【チェックボックス】

チェックボックスモジュールは、画面上に真偽値(オン・オフ)の入力欄を設けるモジュールです。

【アクション】

本項ではチェックボックスモジュールのアクションについて説明します。

【getChecked】

getChecked アクションは、チェックボックスに入っている値を返します。

```
getChecked()
```

【setCaption】

setCaption アクションは、チェックボックスのラベル文字列（キャプション）値を設定します。

```
setCaption(text)
```

text 引数には設定するラベル文字列を与えます。

【setChecked】

setChecked アクションは、チェックボックスの値を設定します。

```
setChecked(value)
```

value 引数には設定する値（true または false）を与えます。チェックボックスの値が変わると後述の【Change】イベントが発生します。

【setDisabled】

setDisabled アクションは、チェックボックスを操作できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setStyle】

setStyle アクションは、チェックボックスのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではチェックボックスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベントAPI】の節を参照してください。

【Blur】

Blur イベントはチェックボックスから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはチェックボックスの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setCheckedアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントはチェックボックスがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはチェックボックスがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはチェックボックスに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはチェックボックス上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはチェックボックス上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ラジオボタン】

ラジオボタンモジュールは、画面上にラジオボタンを設けるモジュールです。複数のラジオボタンに同じ radioname 属性（ラジオボタン名）を付けて 1 つのグループにします。同じグループ内のラジオボタンは 1 つだけが選択された状態（オン）になり残りは選択されていない状態（オフ）になります。グループ単位のラジオボタンの操作については【ラジオボタン群の操作】の節を参照してください。

【アクション】

本項ではラジオボタンモジュールのアクションについて説明します。

【getChecked】

getChecked アクションは、ラジオボタンに入っている値を返します。

```
getChecked()
```

【setCaption】

setCaption アクションは、ラジオボタンのラベル文字列（キャプション）値を設定します。

```
setCaption(text)
```

text 引数には設定するラベル文字列を与えます。

【setChecked】

setChecked アクションは、ラジオボタンの値を設定します。

```
setChecked(value)
```

value 引数には設定する値（true または false）を与えます。ラジオボタンの値が変わると後述の【Change】イベントが発生します。

【setStyle】

setStyle アクションは、ラジオボタンのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではラジオボタンモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはラジオボタンから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはラジオボタンの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setCheckedアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントはラジオボタンがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはラジオボタンがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはラジオボタンに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはラジオボタン上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはラジオボタン上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ラジオボタン群の操作】

スクリーンオブジェクトの radionames プロパティ（this.radionames）は、同じ radioname 属性（ラジオボタン名）を持つラジオボタン群について、現在選択されているラジオボタンのアイテム名やキャプション（caption）を得るためのメソッドを提供します。例えば､radioname が test のラジオボタン群に関して現在選択されているラジオボタンのキャプションを得るには次のようなスクリプトを記述します。

```
var caption = this.radionames.test.getCheckedCaption();
```

また、スクリーンスクリプトの events の中に次のように radioname: {... で始まるオブジェクトの形で radioname 単位の onChange イベントハンドラを記述します。

```
var events = {
    radioname: {
        test_onChange: function(){
            // ここにradionameがtestのラジオボタン群が変化したときの処理を記述する
        },
    },
}
```

同じラジオボタン名（以下、<name> と表記）を持つラジオボタン群について次のメソッドが利用できます。

【getChecked】

getChecked メソッドは選択されたラジオボタンのアイテム名とキャプションを返します。

```
this.radionames.<name>.getChecked()
```

このメソッドは以下のプロパティからなるオブジェクトを返します。

```
name     … アイテム名
caption  … キャプション
```

【getCheckedCaption】

getCheckedCaption メソッドは選択されたラジオボタンのキャプションを返します。

```
this.radionames.<name>.getCheckedCaption()
```

【getCheckedName】

getCheckedName メソッドは選択されたラジオボタンのアイテム名を返します。

```
this.radionames.<name>.getCheckedName()
```

【setCheckedCaption】

setCheckedCaption メソッドは指定されたキャプションを持つラジオボタンを選択された状態にします。

```
this.radionames.<name>.setCheckedCaption(caption)
```

caption 引数には選択状態にするラジオボタンのキャプションを与えます。

【プルダウンリスト】

プルダウンリストモジュールは、画面上にプルダウン方式の選択肢の入力欄を設けるモジュールです。選択肢は次の 2 つの方法で指定できます。

```
a. 選択肢を配列で与える
b. DBテーブル･ビューのカラムから読んだデータを選択肢とする。
```

【アクション】

本項ではプルダウンリストモジュールのアクションについて説明します。

【getSelectedDBValue】

getSelectedDBValue アクションは、プルダウンリストの現在の値に対応する DB テーブル・ビューのデータ行について、指定されたカラムの値を返します。このアクションはプルダウンリストの選択肢を DB テーブル・ビューのカラムから読む場合にのみ利用できます。

```
getSelectedDBValue(colname)
```

colname 引数にはカラム名を指定します。プルダウンリストの選択肢を表示するためにデータベースから読み込んだデータを再利用するので、アプリケーションとサーバの間でデータをやりとりすることなく高速に動作します。

【getValue】

getValue アクションは、プルダウンリストに入っている値を返します。

```
getValue()
```

【reloadDB】

reloadDB アクションは、プルダウンリストの選択肢を DB テーブル･ビューから読み直します。

```
reloadDB()
```

【setDisabled】

setDisabled アクションは、プルダウンリストを操作できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setOptions】

setOptions アクションは、プルダウンリストのオプションを設定します。

```
setOptions(options)
```

options 引数には次の 2 通りの方法でプルダウンリストのオプションを与えます。

```
a.  選択肢を配列で与える
    次のプロパティからなるオブジェクトの配列をoptionsとして指定します。
        name: 表示文字列
        value: 値
b.  DBテーブル･ビューのカラムから読んだデータを選択肢とする
    次のプロパティからなるオブジェクトをoptionsとして指定します。
        modelname: DBテーブル･ビュー名
        columnname: 次のプロパティからなるオブジェクトの配列
            name: 名前のカラム名
            value: 値のカラム名
        num: 最大件数
        useCache: キャッシュ有効
```

【setReadOnly】

setReadOnly アクションは、プルダウンリストを読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStyle】

setStyle アクションは、プルダウンリストのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、プルダウンリストの値を設定します。

```
setValue(value)
```

value 引数には設定する値を与えます。プルダウンリストの値が変わると後述の【Change】イベントが発生します。

【イベント】

本項ではプルダウンリストモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blurイベントはプルダウンリストから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントは選択された値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValue/setOptionsアクションによる変更)または"user"(ユーザによる
        変更)
```

【Click】

Click イベントはプルダウンリストがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはプルダウンリストがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはプルダウンリストに入力フォーカスが移ってきたときに生じます。イ

ベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyDown】

KeyDown イベントはプルダウンリスト上でキーボードのキーが押し下げられたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyPress】

KeyPress イベントはプルダウンリスト上でキーボードのキーが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyPress: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyUp】

KeyUp イベントはプルダウンリスト上で押下されたキーボードのキーが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Load】

Load イベントはプルダウンリストの選択肢が DB テーブル・ビューのカラムから読み込まれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onLoad: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティには読み込まれたデータに関する情報が渡されます。

```
evt.defaultValue … 読み込まれた値カラムの最初の値(デフォルト値)。読み込まれた件数が0
        のときはundefined。
```

【MouseDown】

MouseDown イベントはプルダウンリスト上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはプルダウンリスト上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【コンボボックス】

コンボボックスモジュールは、プルダウンリストとテキストボックスの機能を合わせ持つ入力欄を画面上に設けるモジュールです。プルダウンリストモジュールと同様に、選択肢は次の 2 つの方法で指定できます。

```
a.  選択肢を配列で与える
b.  DBテーブル･ビューのカラムから読んだデータを選択肢とする。
```

また､選択肢にない文字列をテキストボックスモジュールと同様に自由に入力できます。

【アクション】

本項ではコンボボックスモジュールのアクションについて説明します。

【getSelectedDBValue】

getSelectedDBValue アクションは､コンボボックスの現在の値に対応する DB テーブル･ビューのデータ行について、指定されたカラムの値を返します。このアクションはコンボボックスの選択肢を DB テーブル･ビューのカラムから読む場合にのみ利用できます。

```
getSelectedDBValue(colname)
```

colname 引数にはカラム名を指定します。コンボボックスの選択肢を表示するためにデータベースから読み込んだデータを再利用するので、アプリケーションとサーバの間でデータをやりとりすることなく高速に動作します。

【getValue】

getValue アクションは、コンボボックスに入っている値を返します。

```
getValue()
```

【reloadDB】

reloadDB アクションは､コンボボックス内のプルダウンリストの選択肢を DB テーブル･ビューから読み直します。

```
reloadDB()
```

【setOptions】

setOptions アクションは、コンボボックス内のプルダウンリストのオプションを設定します。

```
setOptions(options)
```

options 引数には次の 2 通りの方法でプルダウンリストのオプションを与えます。

```
a.  選択肢を配列で与える
```

```
    次のプロパティからなるオブジェクトの配列をoptionsとして指定します。
        name: 表示文字列
        value: 値
b.  DBテーブル・ビューのカラムから読んだデータを選択肢とする
    次のプロパティからなるオブジェクトをoptionsとして指定します。
        modelname: DBテーブル・ビュー名
        columnname: 次のプロパティからなるオブジェクトの配列
            name: 名前のカラム名
            value: 値のカラム名
        num: 最大件数
        useCache: キャッシュ有効
```

【setReadOnly】

setReadOnly アクションは、コンボボックスを読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStyle】

setStyle アクションは、コンボボックスモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、コンボボックスの値を設定します。

```
setValue(value)
```

value 引数には設定する値を与えます。コンボボックスの値が変わると後述の【Change】イベントが発生します。

【イベント】

本項ではコンボボックスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはコンボボックスから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはコンボボックスの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValue/setOptionsアクションによる変更)または"user"(ユーザによる
        変更)
```

【Click】

Click イベントはコンボボックスがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはコンボボックスがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはコンボボックスに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyDown】

KeyDown イベントはコンボボックス上でキーボードのキーが押し下げられたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyPress】

KeyPress イベントはコンボボックス上でキーボードのキーが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyPress: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyUp】

KeyUp イベントはコンボボックス上で押下されたキーボードのキーが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Load】

Load イベントはコンボボックスの選択肢が DB テーブル・ビューのカラムから読み込まれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onLoad: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティには読み込まれたデータに関する情報が渡されます。

```
evt.defaultValue … 読み込まれた値カラムの最初の値(デフォルト値)。読み込まれた件数が0
        のときはundefined。
```

【MouseDown】

MouseDown イベントはコンボボックス上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはコンボボックス上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【トグルスイッチ】

トグルスイッチモジュールは、押された状態（オン）と押されていない状態（オフ）が交互に切り替わるスイッチを画面上に設けるモジュールです。

【アクション】

本項ではトグルスイッチモジュールのアクションについて説明します。

【getValue】

getValue アクションは、トグルスイッチの現在の状態を表す真偽値（オンなら true、オフなら false）を返します。

```
getValue()
```

【setStyle】

setStyle アクションは、トグルスイッチのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、トグルスイッチの状態を設定します。

```
setValue(value)
```

value 引数には新しい状態を表す真偽値（オンなら true、オフなら false）を与えます。トグルスイッチの状態が変わると後述の【Change】イベントが発生します。

【toggle】

toggle アクションは、トグルスイッチの状態を切り替えます。

```
toggle()
```

toggle アクションを実行すると後述の【Change】イベントが発生します。

【イベント】

本項ではトグルスイッチモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Change】

Change イベントはトグルスイッチの値（状態）が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントはトグルスイッチがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【スライダー】

スライダーモジュールは、画面上にスライド式の数値入力欄を設けるモジュールです。スライダーモジュールでは、入力できる値の範囲（最大値と最小値）を設定できます。

【アクション】

本項ではスライダーモジュールのアクションについて説明します。

【getValue】

getValue アクションは、スライダーの現在の値を返します。

```
getValue()
```

【setDisabled】

setDisabled アクションは、スライダーを操作できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setMax】

setMax アクションは、スライダーで入力できる値の最大値を設定します。

```
setMax(value)
```

value 引数には新しい最大値を与えます。

【setMin】

setMin アクションは、スライダーで入力できる値の最小値を設定します。

```
setMin(value)
```

value 引数には新しい最小値を与えます。

【setReadOnly】

setReadOnly アクションは、スライダーを読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStep】

setStep アクションは、スライダーの入力値の刻み幅を設定します。

```
setStep(value)
```

value 引数には新しい刻み幅を与えます。

【setStyle】

setStyle アクションは、スライダーのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、スライダーの値を設定します。

```
setValue(value)
```

value 引数には新しい値を与えます。スライダーの値が変わると後述の【Change】イベントが発生します。

【イベント】

本項ではスライダーモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはスライダーから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはスライダーの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"（setValueアクションによる変更）または"user"（ユーザによる変更）
```

【Click】

Click イベントはスライダーがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはスライダーに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはスライダー上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはスライダー上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【日時選択】

日時選択モジュールは、画面上に日付入力欄と時刻入力欄を設けるモジュールです。

【アクション】

本項では日時選択モジュールのアクションについて説明します。

【getValue】

getValue アクションは、日時選択モジュールの現在の値を返します。

```
getValue()
```

【setDisabled】

setDisabled アクションは、日時選択モジュールを操作できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setFocus】

setFocus アクションは、日時選択モジュールに入力フォーカスを移します。

```
setFocus()
```

【setHourOptions】

setHourOptions アクションは、時刻入力欄の時間の選択肢を設定します。

```
setHourOptions(array)
```

array 引数には時間の選択肢を数値の配列またはカンマ区切りの数値文字列（例："9,10,11,12,13,14,15,16,17"）で与えます。

【setMaxDate】

setMaxDate アクションは、日付入力欄で選択できる範囲の末尾日を設定します。

```
setMaxDate(date)
```

date 引数には範囲末尾日にセットする日付を Date 型オブジェクト（または Date 型に変換できる日付文字列）で与えます。

【setMinDate】

setMinDate アクションは、日付入力欄で選択できる範囲の先頭日を設定します。

```
setMinDate(date)
```

date引数には範囲先頭日にセットする日付をDate型オブジェクト（またはDate型に変換できる日付文字列）で与えます。

【setMinuteOptions】

setMinuteOptions アクションは、時刻入力欄の分の選択肢を設定します。

```
setMinuteOptions(array)
```

array 引数には分の選択肢を数値の配列またはカンマ区切りの数値文字列（例："0,15,30,45"）で与えます。

【setReadOnly】

setReadOnly アクションは、日時選択モジュールを読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStyle】

setStyle アクションは、日時選択モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、日時選択モジュールの値を設定します。

```
setValue(value)
```

value 引数には新しい値（日付文字列）を与えます。値として有効な日付文字列については【フィルタ API】の節（YMD フィルタの項）を参照してください。日時選択モジュールの値が変わると後述の【Change】イベントが発生します。

【setUseTime】

setUseTime アクションは、日時選択モジュールの時刻入力欄を用いるかどうかを指定します。

```
setUseTime(value)
```

value 引数には、時刻を用いる場合は以下のいずれかの文字列を与えます。

```
hms      … 時分秒を使用
hm       … 時分のみを使用
```

時刻入力欄を使用しない場合は空文字列を与えます。

【イベント】

本項では日時選択モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントは日時選択モジュールから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントは日時選択モジュールの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Focus】

Focus イベントは日時選択モジュールに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【色選択】

色選択モジュールは、画面上に色の入力欄を設けるモジュールです。色は #xxxxxx の形の文字列(xxxxxx は 6 桁の 16 進数)で表されます。入力欄をクリックすると色を対話的に選択できるダイアログが開きます。

【アクション】

本項では色選択モジュールのアクションについて説明します。

【getValue】

getValue アクションは、色選択モジュールの現在の値を返します。

```
getValue()
```

【setDisabled】

setDisabled アクションは、色選択モジュールを操作できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setReadOnly】

setReadOnly アクションは、色選択モジュールを読み取り専用に設定します。

```
setReadOnly(value)
```

value 引数には真偽値を与えます。value が真のときは読み取り専用になり、それ以外のときは読み取り専用状態が解除されます。

【setStyle】

setStyle アクションは、色選択モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、色選択モジュールの値を設定します。

```
setValue(value)
```

value 引数には新しい値を与えます。色選択モジュールの値が変わると後述の【Change】イベントが発生します。

【イベント】

本項では色選択モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントは色選択モジュールから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントは色選択モジュールの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントは色選択モジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントは色選択モジュールに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DB レコードセレクタ】

DB レコードセレクタモジュールは、DB テーブルモジュールと連動して DB テーブル・ビューの表示を制御するためのボタン類を画面上に設けるモジュールです。DB レコードセレクタは、DB テーブルモジュールに表示される一連のレコードの最初のレコード番号を値として保持します。

【アクション】

本項では DB レコードセレクタモジュールのアクションについて説明します。

【getValue】

getValue アクションは、DB レコードセレクタの現在の値を返します。

```
getValue()
```

【moveFirst】

moveFirst アクションは、DB レコードセレクタの値を DB テーブル・ビューの最初のレコードに設定します。

```
moveFirst()
```

moveFirst アクションを実行すると後述の【Change】イベントが発生します。

【moveLast】

moveLast アクションは、DB レコードセレクタの値を DB テーブル・ビューの最後のレコードに設定します。

```
moveLast()
```

moveLast アクションを実行すると後述の【Change】イベントが発生します。

【moveNext】

moveNext アクションは、DB レコードセレクタの値を次のレコードに設定します。

```
moveNext()
```

moveNext アクションを実行すると後述の【Change】イベントが発生します。

【movePrev】

movePrev アクションは、DB レコードセレクタの値を 1 つ前のレコードに設定します。

```
movePrev()
```

movePrev アクションを実行すると後述の【Change】イベントが発生します。

【searchDone】

searchDone アクションは、DB レコードセレクタのキーワード検索の状態を更新します。

```
searchDone(offset)
```

offset 引数には DB テーブルモジュールの SearchDone イベントを通じてアプリケーション側に渡されてくるキーワード検索結果のレコード番号を与えます。このレコード番号は DB レコードセレクタの検索ボタンを繰り返し押して、キーワードを含むレコードを順次検索するという動作を実現するために用いられます。

【setMax】

setMax アクションは、DB レコードセレクタの値の最大値（DB テーブル・ビューのレコード数）を設定します。

```
setMax(max)
```

max 引数には最大レコード番号を与えます。setMax アクションを実行すると後述の【Change】イベントが発生します。

【setStyle】

setStyle アクションは、DB レコードセレクタモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、DB レコードセレクタの値を設定します。

```
setValue(value)
```

value 引数には新しい値を与えます。setValue アクションを実行すると後述の【Change】イベントが発生します。

【useSearch】

useSearch アクションは、DB レコードセレクタのキーワード検索機能を有効または無効にします。

```
useSearch(value)
```

value 引数には真偽値を与えます。value が真のときはキーワード検索機能が有効状態になり、キーワード入力欄が表示されます。それ以外のときはキーワード検索機能が無効状態になり、キーワード入力欄が非表示になります。

【イベント】

本項では DB レコードセレクタモジュールから発生するイベントについて説明します。

【Change】

Change イベントは DB レコードセレクタモジュールの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Search】

Search イベントは DB レコードセレクタモジュールの検索ボタンが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onSearch: function(data){ ... }
```

イベントハンドラの data 引数には以下のプロパティから成るオブジェクトが渡されます。

```
data.keyword … 検索キーワード
data.searchPos … 検索開始位置(最初のレコードが0)
```

たとえば DB レコードセレクタモジュール RECORDSELECTOR1 と DB テーブルモジュール DATABASETABLE1 を連携させてキーワード検索を実現するには以下のようなイベントハンドラを記述します。

```
RECORDSELECTOR1_onSearch: function(data){
    this.items.DATABASETABLE1.search(data.keyword, data.searchPos);
},
DATABASETABLE1_onSearchDone : function(data){
    this.items.RECORDSELECTOR1.searchDone(data.offset);
},
```

検索結果（検索キーワードを含むレコードの番号）は DB テーブルモジュールの SearchDone イベントを通じてアプリケーション側に返されます。これを受け取って DB レコードセレクタの searchDone アクションに渡します。

【タブ】

タブモジュールは、ひとつのスクリーンコンテナで複数のスクリーン（タブ）を切り替えて表示ためのボタン群を画面上に配置するためのモジュールです。

【アクション】

本項ではタブモジュールのアクションについて説明します。

【getSelected】

getSelected アクションは現在選択されているタブ名を返します。

```
getSelected()
```

【setItems】

setItems アクションはタブ切り替えボタンに表示される名前（タブ名）を設定します。

```
setItems(items)
```

items 引数には [{"name": " タブ名 1"}, {"name": " タブ名 2"}, …] の形式の JSON 文字列を与えます。第 N 要素の name プロパティの値が N 番目のタブ名として用いられます。

【setSelected】

setSelected アクションは、タブモジュールの表示するタブを設定します。

```
setSelected(name)
```

name 引数には表示するタブの名前を文字列で与えます。

【setStyle】

setStyle アクションは、タブモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではタブモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Change】

Change イベントはタブモジュールの表示するタブが変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.tab　… 選択されたタブを表すオブジェクト
evt.cause … "set"（setSelectedアクションによる変更）または"user"（ユーザによる変更）
```

【Click】

Click イベントはタブモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数evtにはイベントオブジェクトが渡されます。また、evt.tabにはクリックされたタブ切り替えボタンに関する情報を格納した以下のプロパティから成るオブジェクトが渡されます。

```
evt.tab.name　…　クリックされたタブ名
```

タブモジュールTAB1とスクリーンコンテナモジュールSCREENCONTAINER1を連携させて複数のスクリーンを切り替えるには次のようなイベントハンドラを記述します。

```
TAB1_onClick: function(evt){
    var tab = evt.tab;
    this.items.SCREENCONTAINER1.setScreen(tab.name);
}
```

【メニュー】

メニューモジュールは、プルダウン方式のメニューを表示するためのメニューボタン群を画面上に配置するためのモジュールです。

【アクション】

本項ではメニューモジュールのアクションについて説明します。

【open】

open アクションは指定されたメニューを開きます。

```
open(name)
```

name 引数には開くメニュー名を文字列で与えます。

【setStyle】

setStyle アクションは、メニューモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではメニューモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはメニューモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt.menu にはクリックされたメニューボタンに関する情報を格納した以下のプロパティから成るオブジェクトが渡されます。

```
evt.menu.name  …  クリックされたメニュー名
```

【ファイル選択】

ファイル選択モジュールは、サーバにアップロードするファイルを選択するための入力欄を画面上に設けるモジュールです。

【アクション】

本項ではファイル選択モジュールのアクションについて説明します。

【clear】

clear アクションは、ファイル選択モジュールをファイルが選択されていない状態にします。

```
clear()
```

【getFiles】

getFiles アクションは、選択されたファイルの集合を表す FileList オブジェクトを返します。FileListオブジェクトの詳細についてはHTML5 File APIのドキュメントを参照してください。

```
getFiles()
```

【getValue】

getValue アクションは、ファイル選択モジュールの現在の値を返します。

```
getValue()
```

【readFile】

readFile アクションは、選択されたファイルの内容を読み取ります。

```
readFile(options, callback)
```

options 引数には以下のプロパティからなるオブジェクトを与えます。

```
index    … 選択されたファイルのうち、内容を読み取るファイルの番号（先頭が0）
encode   … ファイルの文字コード（デフォルトは"UTF-8"）
```

callback 引数にはコールバック関数を与えます。ファイル内容の読み取りが失敗した場合、第 1 引数にエラーメッセージ文字列が渡されます。成功した場合はコールバック関数の第 1 引数には null、第 2 引数にはファイルの内容が渡されます。

【setDisabled】

setDisabled アクションは、ファイル選択モジュールを操作できない無効状態に設定します。

```
setDisabled(value)
```

value 引数には真偽値を与えます。value が真のときは無効状態になり、それ以外のときは無効状態が解除されます。

【setStyle】

setStyle アクションは、ファイル選択モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【upload】

upload アクションは、選択されたファイルをサーバにアップロードします。

```
upload(dir, options, callback)
```

dir 引数にはサーバ上の保存先ディレクトリを文字列で与えます。options 引数は現在は使用されません（空オブジェクト {} を与えてください）。callback 引数にはコールバック関数を与えます。ファイル内容の読み取りが失敗した場合、第 1 引数に Error オブジェクトが渡されます。成功した場合はコールバック関数の第 1 引数には null、第 2 引数にはアップロード結果が渡されます。

【イベント】

本項ではファイル選択モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはファイル選択モジュールから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Change】

Change イベントはファイル選択モジュールの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
```

【Click】

Click イベントはファイル選択モジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはファイル選択モジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはファイル選択モジュールに入力フォーカスが移ってきたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyDown】

KeyDown イベントはファイル選択モジュール上でキーボードのキーが押し下げられたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyPress】

KeyPress イベントはファイル選択モジュール上でキーボードのキーが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyPress: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyUp】

KeyUp イベントはファイル選択モジュール上で押下されたキーボードのキーが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはファイル選択モジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはファイル選択モジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【画像】

画像モジュールは、画面上に画像を表示するためのモジュールです。

【アクション】

本項では画像モジュールのアクションについて説明します。

【getAlt】

getAlt アクションは、画像モジュールが表示する代替テキストを得ます。

```
getAlt()
```

【getSrc】

getSrc アクションは、画像モジュールが表示する画像ファイルの URL を得ます。

```
getSrc()
```

【getValue】

getValue アクションは、画像モジュールが表示する画像ファイルの URL を得ます。

```
getValue()
```

【setAlt】

setAlt アクションは、画像モジュールが表示する代替テキストを設定します。

```
setAlt(value)
```

value 引数には新しい代替テキストを文字列で与えます。

【setSrc】

setSrc アクションは、画像モジュールが表示する画像ファイルの URL を設定します。

```
setSrc(value)
```

value 引数には新しい画像ファイルの URL を文字列で与えます。

【setStyle】

setStyle アクションは、画像モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、画像モジュールが表示する画像ファイルの URL を設定します。

```
setValue(value)
```

value 引数には新しい画像ファイルの URL を文字列で与えます。

【イベント】

本項では画像モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントは画像モジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントは画像モジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Error】

Error イベントは指定された画像ファイルの読み込みに失敗したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onError: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントは画像モジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントは画像モジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【文字列】

文字列モジュールは、画面上に文字列を表示するためのモジュールです。文字列モジュールは表示している文字列を値として保持します。

【アクション】

本項では文字列モジュールのアクションについて説明します。

【getValue】

getValue アクションは、文字列モジュールが表示する値（文字列）を返します。

```
getValue()
```

【setFilter】

setFilter アクションは、文字列モジュールに適用するフィルタを設定します。

```
setFilter(filters)
```

filters 引数にはフィルタ名を文字列で与えます。フィルタ名はカンマで区切って複数指定できます。フィルタの詳細については後述の【フィルタ API】の節を参照してください。

【setStyle】

setStyle アクションは、文字列モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、文字列モジュールが表示する値（文字列）を設定します。

```
setValue(value)
```

value 引数には新しい値を与えます。文字列モジュールの値が変わると後述の【Change】イベントが発生します。

【イベント】

本項では文字列モジュールから発生するイベントについて説明します。

【Change】

Change イベントは文字列モジュールの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.cause … "set"(setValueアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントは文字列モジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントは文字列モジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントは文字列モジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントは文字列モジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【数式】

数式モジュールは自らの数値データを他のモジュールの数値データから自動的に計算して表示するためのモジュールです。計算内容は数式ハンドラとして JavaScript で記述します。数式ハンドラについては【数式 API】の章を参照してください。

【アクション】

本項では数式モジュールのアクションについて説明します。

【getValue】

getValue アクションは数式モジュールの現在の値を返します。

```
getValue()
```

【setStyle】

setStyle アクションは、数式モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項では数式モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Change】

Change イベントは数式モジュールの値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(){ ... }
```

【Click】

Click イベントは数式モジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントは数式モジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントは数式モジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントは数式モジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【リスト】

リストモジュールは複数の項目を選択できる選択肢入力欄を画面上に設けるモジュールです。

【アクション】

本項ではリストモジュールのアクションについて説明します。

【getSelectedDBValue】

getSelectedDBValue アクションは、選択されたリスト項目に対応する DB テーブル・ビューのデータ行について、指定されたカラムの値を返します。このアクションはリストモジュールの選択肢を DB テーブル・ビューのカラムから読む場合にのみ利用できます。

```
getSelectedDBValue(colname)
```

colname 引数にはカラム名を指定します。リストモジュールの選択肢を表示するためにデータベースから読み込んだデータを再利用するので、アプリケーションとサーバの間でデータをやりとりすることなく高速に動作します。

【getValue】

getValue アクションは選択されたリスト項目の値を文字列配列として返します。

```
getValue()
```

【getSelection】

getSelection アクションは選択されたリスト項目の番号（先頭が 0）を配列で返します。

```
getSelection()
```

【reloadDB】

reloadDB アクションは、リストモジュールの選択肢を DB テーブル・ビューから読み直します。

```
reloadDB()
```

【setOptions】

setOptions アクションはリストモジュールの動作オプションを設定します。

```
setOptions(options)
```

options 引数には次の 2 通りの方法でオプションを与えます。

```
a.　選択肢を配列で与える
```

```
    次のプロパティからなるオブジェクトの配列をoptionsとして指定します。
        name: 表示文字列
        value: 値
b.  DBテーブル・ビューのカラムから読んだデータを選択肢とする
    次のプロパティからなるオブジェクトをoptionsとして指定します。
        modelname: DBテーブル・ビュー名
        columnname: 次のプロパティからなるオブジェクトの配列
            name: 名前のカラム名
            value: 値のカラム名
        num: 最大件数
```

【setSelection】

setSelection アクションは指定されたリスト項目を選択された状態に設定します。

```
setSelection(index)
```

index 引数には選択状態にする項目の番号（先頭が 0）の配列を与えます。

【setStyle】

setStyle アクションは、リストモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではリストモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Change】

Change イベントはリストモジュールの動作オプションの変更により値が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … 変化後の値
evt.oldValue … 変化前の値
evt.cause … "set"(setOptionsアクションによる変更)または"user"(ユーザによる変更)
```

【Click】

Click イベントはリストモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティにはクリックされたリスト項目の情報が渡されます。

```
evt.index … クリックされたリスト項目の番号(最初のリスト項目が0)
```

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはリストモジュールがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Load】

Load イベントはリストモジュールの選択肢が DB テーブル・ビューのカラムから読み込まれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onLoad: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはリストモジュール上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはリストモジュール上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【箇条書き】

箇条書きモジュールは画面上に箇条書きのリストを表示するモジュールです。

【アクション】

本項では箇条書きモジュールのアクションについて説明します。

【setItems】

setItems アクションは表示するリスト項目を設定します。

```
setItems(items)
```

items 引数にはリスト項目を文字列の配列で与えます。

【setStyle】

setStyle アクションは、箇条書きモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項では箇条書きモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントは箇条書きモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティにはクリックされたリスト項目の情報が渡されます。

```
evt.index … クリックされたリスト項目の番号(最初のリスト項目が0)
```

【テーブル】

テーブルモジュールは行と列から成る表（テーブル）形式のデータを表示するためのモジュールです。テーブルには列ラベルを示すヘッダ行を付与することができます。

【アクション】

本項ではテーブルモジュールのアクションについて説明します。

【setCellStyle】

setCellStyle アクションはセル毎にスタイルを設定します。

```
setCellStyle(func)
```

func 引数には関数を与えます。この関数はテーブルの各セルに対して呼び出されます。関数の呼び出し時には 3 つの引数が渡されます。関数の引数リストを (rowIndex, colIndex, content) とすると、rowIndex 引数はセルの行番号、colIndex 引数はセルの列番号、content 引数はセルの内容文字列（フィルタ適用前の内容文字列）を表します。行番号と列番号は 0 から始まります。この関数はスタイルオブジェクトまたは {outer: セルスタイル , inner: 内容スタイル } の形式のオブジェクトを返さなければなりません。

【setColStyle】

setColStyle アクションはカラムのスタイルを設定します。

```
setColStyle(style)
```

style 引数には（a）スタイルオブジェクト、（b）スタイルオブジェクトの配列、（c）{outer: セルスタイル , inner: 内容スタイル } の形式のオブジェクト、または（d）（c）の形式のオブジェクトの配列を与えます。（a）および（c）の場合は全カラムに同じスタイルを適用します。（b）および（d）の場合は N 番目の要素で与えられたスタイルが N 番目のカラムに適用されます。outer はセル（<td> 要素）に、inner はセルの内容に適用されます。

【setColWidth】

setColWidth アクションはテーブルの表示上の列幅を設定します。

```
setColWidth(width)
```

width 引数には列幅を（a）単独の値、（b）値の配列、（c）{outer: セル幅 , inner: 内容幅 } の形式のオブジェクト、または（d）（c）の形式のオブジェクトの配列で与えます。（a）および（c）の場合は全カラムが同じ列幅になります。（b）および（d）の場合は N 番目の要素で与えられた設定が N 番目のカラムの列幅となります。outer はセル（<td> 要素）に、inner はセルの内容に適用されます。列幅はピクセル数、パーセント値、「auto」を指定できます。パーセント指定は、outer ならテーブル幅に対する割合、inner ならセル幅に対する割合を意味します。

【setFilter】

setFilter アクションはテーブルのセルの内容に適用するフィルタを指定します。

```
setFilter(filters)
```

filters 引数にはカラム名とフィルタ名の対を以下の形式のオブジェクトで与えます。フィルタ名はカンマで区切って複数指定できます。

```
{"カラム名1": フィルタ名1, "カラム名2": フィルタ名2, …}
```

フィルタの詳細については後述の【フィルタ API】の節を参照してください。

【setHeader】

setHeader アクションはヘッダ行を設定します。

```
setHeader(headers)
```

headers 引数には（a）カラム名文字列の配列、または（b）以下のプロパティから成るオブジェクトの配列を与えます。

```
name      … カラム名
displayName … 表示名
alignment … 配置。"left"(左寄せ)、"center"(中央)、"right"(右寄せ)のいずれか
```

【setStyle】

setStyle アクションは、テーブルモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数には（a）スタイルオブジェクト、または（b）{outer: アイテムスタイル , inner: テーブルスタイル } を与えます。(a) の場合はアイテム（<table> 要素の外側の 要素）にスタイルが適用されます。(b) の場合、outer で与えたスタイルはアイテムに、inner で与えたスタイルはテーブル（<table> 要素）にそれぞれ適用されます。

【setTable】

setTable アクションはテーブルデータを設定します。

```
setTable(table)
```

table 引数にはテーブルデータを（a）文字列配列の配列、または（b）{ カラム名 : 値 , … } の形式のオブジェクトの配列で与えます。(a) の場合は上述の setHeader アクションでヘッダデータを設定しなければヘッダ行は表示されません。また、(b) の場合は setHeader アクションでヘッダデータを設定しなければテーブル自体が表示されません。

【DB テーブル】

DB テーブルモジュールは DB テーブルに格納されたデータを画面上に表示するためのモジュールです。

【アクション】

本項では DB テーブルモジュールのアクションについて説明します。

【getCount】

getCount アクションは現在設定されている絞り込み条件の下で表示されるレコードの数を問い合わせます。

```
getCount()
```

問い合わせ結果のレコード数は ChangeCount イベント（後述）を通じてアプリケーション側に返されます。

【initialize】

initialize アクションは DB テーブルモジュールの表示内容を初期化します。

```
initialize(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
table    … DBテーブル・ビューのモデルオブジェクト（必須）
headers  … 表示するカラム名の配列。配列要素は文字列または以下のプロパティから成るオブ
           ジェクト：
            name      … カラム名
            displayName … 表示名
            alignment … 配置。"left"（左寄せ）、"center"（中央）、"right"（右寄せ）のいずれか
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照。
order    … 並べ替え。後述の【並べ替え】の節を参照。
num      … 表示件数
offset   … 先頭レコード番号（最初のレコードが0）
useCache … キャッシュ利用フラグ。真ならキャッシュを用います。
crossOptions … クロス集計オプション。クロス集計モジュールのUpdateイベントを通じて渡さ
           れてくるcrossOptionsプロパティの値を与えます。後述の【クロス集計オプション】の節
           を参照。
```

【outputData】

outputData アクションは DB テーブルモジュールの表示内容を外部ファイルとして出力します。

```
outputData(options)
```

options 引数にはデータ出力オプションを以下のプロパティから成るオブジェクトで与

えます。

```
format   … 出力ファイル形式。現在は "xlsx"(Microsoft Excel形式)のみサポート。
fileName … 出力ファイル名。
```

データ出力が成功した場合は OutputDataCompleted イベント（後述）を通じて出力ファイルをダウンロードするための URL がアプリケーション側に返されます。出力が失敗した場合は OutputDataFailed イベント（後述）を通じてエラーメッセージが返されます。

【search】

search アクションは DB テーブル内のデータに対するキーワード検索を実行します。

```
search(keyword, searchPos)
```

keyword 引数には検索キーワードを文字列で与えます。searchPos 引数にはキーワード検索の開始位置となるレコード番号を与えます（最初のレコードが 0)。検索結果は SearchDone イベント（後述）を通じてアプリケーション側に返されます。

【setCellStyle】

setCellStyle アクションはセル毎にスタイルを設定します。

```
setCellStyle(func)
```

func 引数には関数を与えます。この関数は DB テーブルの各セルに対して呼び出されます。関数の呼び出し時には 3 つの引数が渡されます。関数の引数リストを (rowIndex, colIndex, content) とすると、rowIndex 引数はセルの行番号、colIndex 引数はセルの列番号、content 引数はセルの内容文字列（フィルタ適用前の内容文字列）を表します。行番号と列番号は 0 から始まります。この関数はスタイルオブジェクトまたは {outer: セルスタイル , inner: 内容スタイル } の形式のオブジェクトを返さなければなりません。

【setColStyle】

setColStyle アクションはカラムのスタイルを設定します。

```
setColStyle(style)
```

style 引数には（a）スタイルオブジェクト、(b) スタイルオブジェクトの配列、(c) {outer: セルスタイル , inner: 内容スタイル } の形式のオブジェクト、または (d) (c) の形式のオブジェクトの配列を与えます。(a) および (c) の場合は全カラムに同じスタイルを適用します。(b) および (d) の場合は N 番目の要素で与えられたスタイルが N 番目のカラムに適用されます。outer はセル（<td> 要素）に、inner はセルの内容に適用されます。

【setColWidth】

setColWidth アクションは DB テーブルの表示上の列幅を設定します。

```
setColWidth(width)
```

width 引数には列幅を（a）単独の値、(b) 値の配列、(c) {outer: セル幅 , inner: 内容幅 }

の形式のオブジェクト、または（d）（c）の形式のオブジェクトの配列で与えます。（a）および（c）の場合は全カラムが同じ列幅になります。（b）および（d）の場合は N 番目の要素で与えられた設定が N 番目のカラムの列幅となります。outer はセル（<td> 要素）に、inner はセルの内容に適用されます。列幅はピクセル数、パーセント値、「auto」を指定できます。パーセント指定は、outer ならテーブル幅に対する割合、inner ならセル幅に対する割合を意味します。

【setFilter】

setFilter アクションは DB テーブルから読み取ったカラムの値に適用するフィルタを指定します。

```
setFilter(filters)
```

filters 引数にはカラム名とフィルタ名の対を以下の形式のオブジェクトで与えます。フィルタ名はカンマで区切って複数指定できます。

```
{"カラム名1": フィルタ名1, "カラム名2": フィルタ名2, …}
```

フィルタの詳細については後述の【フィルタ API】の節を参照してください。

【setNum】

setNum アクションは表示するレコードの数を設定します。

```
setNum(num)
```

num 引数には表示レコード数を与えます。

【setOffset】

setOffset アクションは先頭レコード番号を設定します。

```
setOffset(offset)
```

offset 引数にはレコード番号を与えます（最初のレコードが 0）。

【setOrder】

setOrder アクションは表示レコードのソート順を設定します。

```
setOrder(order)
```

order 引数には並べ替えの方法を記述するオブジェクトを与えます。後述の【並べ替え】の節を参照してください。

【setStyle】

setStyle アクションは DB テーブルモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数には（a）スタイルオブジェクト、または（b）{outer: アイテムスタイル , inner:

テーブルスタイル } を与えます。(a) の場合はテーブル (<table> 要素) にスタイルが適用されます。(b)の場合、outer で与えたスタイルはアイテム(<table> 要素の外側の 要素) に、inner で与えたスタイルはテーブルにそれぞれ適用されます。

【setWhere】

setWhere アクションは表示レコードの絞り込み条件を設定します。

```
setWhere(where)
```

where 引数には絞り込み条件を与えます。後述の【絞り込み条件】の節を参照してください。

【イベント】

本項では DB テーブルモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【ChangeCount】

ChangeCount は絞り込み条件が変更されたとき、および getCount アクションが実行されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChangeCount: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.value … レコード数
```

【Click】

Click イベントは DB テーブルモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティにクリックされたカラムに関する情報が渡されます。

```
evt.row  … クリックされた行番号
evt.col  … クリックされた列番号
evt.data … クリックされたレコード({カラム名: 値, …}の形式のオブジェクト)
```

【LoadData】

LoadData イベントは DB テーブルから表示データを読み取り終えた終えたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onLoadData: function(){ ... }
```

【OutputDataCompleted】

OutputDataCompleted イベントは outputData アクションによるファイル出力が成功したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onOutputDataCompleted: function(data){ ... }
```

data 引数には出力ファイルに関する情報が以下のプロパティを持つオブジェクトとして渡されます。

```
data.url … 出力ファイルのダウンロードURL
```

【OutputDataFailed】

OutputDataFailed イベントは outputData アクションによるファイル出力が失敗したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onOutputDataFailed: function(error){ ... }
```

error 引数にはエラーに関する情報が以下のプロパティを持つオブジェクトで渡されます。

```
error.message … エラーメッセージ
```

【SearchDone】

SearchDone イベントは search アクションによるキーワード検索が完了したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onSearchDone: function(data){ ... }
```

data 引数には検索結果に関する情報が以下のプロパティを持つオブジェクトとして渡されます。

```
data.offset … 検索キーワードが見つかったレコードの番号(最初のレコードが0)
```

DB テーブルモジュールと DB レコードセレクタモジュールの検索機能の連携方法については【DB レコードセレクタ】の節を参照してください。

【スピン】

スピンモジュールは、回転する矢印を画面上にアニメーション表示するためのモジュールです。表示と非表示の 2 つの状態に切り替えられます。表示状態のときは回転アニメーションが繰り返されます。アプリケーションが時間のかかる処理をしている待ち時間などに用います。スピンモジュールは現在の表示状態を示す真偽値（表示中なら true、非表示なら false）を値として保持します。

【アクション】

本項ではスピンモジュールのアクションについて説明します。

【getStatus】

getStatus アクションは、スピンモジュールの表示状態を文字列（表示中なら "visible"、非表示なら "hidden"）を返します。

```
getStatus()
```

【getValue】

getValue アクションは、スピンモジュールの現在の値（表示中なら true、非表示なら false）を返します。

```
getValue()
```

【hide】

hide アクションは、スピンモジュールを非表示にします。

```
hide()
```

【setStyle】

setStyle アクションは、スピンモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションは、スピンモジュールの値を設定します。

```
setValue(value)
```

value 引数には新しい値を与えます。

【show】

show アクションは、スピンモジュールを表示します。

```
show()
```

【toggle】

toggle アクションは、スピンモジュールの表示・非表示の状態を入れ替えます。

```
toggle()
```

【イベント】

本項ではスピンモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはスピンモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【プログレスバー】

プログレスバーモジュールは時間のかかる処理が全体の何パーセントまで進んだか（進捗率）を視覚的に示すためのモジュールです。

【アクション】

本項ではプログレスバーモジュールのアクションについて説明します。

【getValue】

getValue アクションはプログレスバーの進捗率を返します。

```
getValue()
```

【setStyle】

setStyle アクションはプログレスバーのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setValue】

setValue アクションはプログレスバーの進捗率を設定します。

```
setValue(value)
```

value 引数には進捗率を 0 〜 100 の数値で与えます。

【イベント】

本項ではプログレスバーモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはプログレスバーがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはプログレスバー上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはプログレスバー上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【カレンダー】

カレンダーモジュールは画面上にカレンダーを表示するモジュールです。カレンダーには日付を指定してテキスト項目を挿入することができます。

【アクション】

本項ではカレンダーモジュールのアクションについて説明します。

【addText】

addText アクションはカレンダーにテキスト項目を追加表示します。

```
addText(date, text)
```

date 引数には挿入先の日付を Date 型で与えます。text 引数には表示する文字列を指定します。

【clearText】

clearText アクションは指定の日付に付与されているテキスト項目をすべて削除します。

```
clearText(date)
```

date 引数には日付を Date 型で与えます。

【delText】

delText アクションは指定の日付に付与されているテキスト項目をひとつ削除します。

```
delText(date, index)
```

date 引数には日付を Date 型で与えます。index 引数には削除するテキスト項目の番号を与えます（最初のテキスト項目が 0）。

【setStyle】

setStyle アクションは、カレンダーモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【setYearMonth】

setYearMonth アクションはカレンダーの表示年・月を設定します。

```
setYearMonth(year, month)
```

year 引数と month 引数には表示する年と月をそれぞれ数値で与えます。setYearMonth アクションを実行すると後述の【Change】イベントが発生します。

【textLength】

textLength アクションは指定された日付に付与されているテキスト項目の数を返します。

```
textLength(date)
```

date 引数には日付を Date 型で与えます。

【イベント】

本項ではカレンダーモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Change】

Change イベントはカレンダーモジュールが表示する年または月が変化したときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onChange: function(data){ ... }
```

イベントハンドラの引数 data には以下のプロパティから成るオブジェクトが渡されます。

```
data.year … 変更後の年
data.month … 変更後の月
data.cause … "set"（setYearMonthアクションによる変更）または"user"（ユーザによる変更）
```

【Click】

Click イベントはカレンダーモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、クリックした日付が evt.date プロパティに Date 型で渡されます。

【タイムテーブル】

タイムテーブルモジュールは画面上に時間割表を表示するモジュールです。時間割表には開始・終了日時を指定してテキスト項目を挿入することができます。

【アクション】

本項ではタイムテーブルモジュールのアクションについて説明します。

【addText】

addText アクションはタイムテーブルにテキスト項目を追加表示します。

```
addText(start, end, text, bgcolor)
```

start 引数と end 引数には開始日時と終了日時をそれぞれ Date 型オブジェクト（または Date 型に変換できる日時文字列）で与えます。text 引数には表示文字列、bgcolor 引数には表示文字列の背景色を指定します。text 引数は文字列に加えて次の形式のオブジェクトを受け付けます。

```
{ text: 表示文字列, popup: ポップアップ文字列 }
```

popup プロパティを省略すると text プロパティの値がポップアップ文字列として用いられます。このオブジェクトには他のプロパティも自由に含めてよく、Click イベントハンドラに渡される evt.text プロパティには上記のオブジェクトがそのまま渡されます。たとえば text 引数の値を

```
{ id: データ番号, text: 表示文字列, popup: ポップアップ文字列 }
```

としておけば、Click イベントハンドラではデータ番号を evt.text.id で受け取れます。

【clearText】

clearText アクションはすべてのテキスト項目を削除します。

```
clearTex()
```

【delText】

delText アクションはテキスト項目をひとつ削除します。

```
delText(index)
```

index 引数には削除するテキスト項目の番号を与えます（最初のテキスト項目が 0）。

【setDays】

setDays アクションは開始日からの表示日数を設定します。

```
setDays(days)
```

days 引数には日数を数値で与えます。

【setStartDate】

setStartDate アクションはタイムテーブルの表示開始日を設定します。

```
setStartDate(startdate)
```

startdate 引数には表示開始日を Date 型オブジェクトまたは日付文字列(" 年 / 月 / 日 " または " 年 - 月 - 日 ") で与えます。

【setStyle】

setStyle アクションは、タイムテーブルモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【textLength】

textLength アクションはテキスト項目の数を返します。

```
textLength()
```

【イベント】

本項ではタイムテーブルモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントはタイムテーブルモジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、タイムテーブル内のどこをクリックしたかに応じて evt オブジェクトの以下のプロパティに情報が渡されます。

```
a. テキスト項目をクリックした場合
    evt.data … クリックしたデータ項目。以下のプロパティから成るオブジェクト:
        start    … 開始日時(Date型)
        end      … 終了日時(Date型)
        text     … 表示文字列
        bgcolor  … 色
b. テキスト項目以外をクリックした場合
    evt.datetime … クリックした日時(Date型)
```

【クロス集計】

クロス集計モジュールは DB テーブル・ビューのクロス集計を行なうための各種オプションを画面上に表示するモジュールです。クロス集計した結果の表示には【DB テーブル】モジュールを用います。

【アクション】

本項ではクロス集計モジュールのアクションについて説明します。

【initialize】

initialize アクションはクロス集計モジュールを初期化します。

```
initialize(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
models   … モデル情報オブジェクト(this.models)
num      … 表示するレコード数
offset   … 先頭レコード番号(最初のレコードが0)
```

【setStyle】

setStyle アクションはクロス集計モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではクロス集計モジュールから発生するイベントについて説明します。

【Update】

Update イベントはクロス集計モジュールの更新ボタンが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onUpdate: function(data){ ... }
```

data 引数には以下のプロパティから成るオブジェクトが渡されます。

```
table     … クロス集計するDBテーブル・ビューのモデルオブジェクト
crossOptions … クロス集計オプション。後述の【クロス集計オプション】の節を参照してください。
```

これらのプロパティは DB テーブルモジュールの【initialize】アクションのオプションとして利用できます。たとえばクロス集計モジュール CROSS1 から渡されたクロス集計オプションを使って DB テーブルモジュール DATABASETABLE1 の表示内容を初期化

するには次のようなイベントハンドラを作成します。

```
CROSS1_onUpdate: function(data){
    var options = {
        table: data.table,
        crossOptions: data.crossOptions,
    };
    this.items.DATABASETABLE1.initialize(options);
}
```

【地図】

地図モジュールは画面上に地図を表示するモジュールです。地図上には緯度・経度で指定した地点にマーカーを設けて情報を表示することができます。

【アクション】

本項では地図モジュールのアクションについて説明します。

【addMarker】

addMarker アクションは緯度・経度で表された地点にマーカーを追加表示します。マーカーにはポップアップ表示されるマークアップ付きテキスト（HTML）を付与できます。

```
addMarker(data)
```

data 引数には以下のプロパティから成るオブジェクトを与えます。

```
lat     … マーカーを設ける地点の緯度
lng     … マーカーを設ける地点の経度
popup   … ポップアップ表示するHTML文字列
```

【clearMarkers】

clearMarkers アクションはすべてのマーカーを削除します。

```
clearMarkers()
```

【delMarker】

delMarker アクションはマーカーをひとつ削除します。

```
delMarker(index)
```

index 引数には削除するマーカーの番号を与えます（最初のマーカーが 0）。

【markersLength】

markersLength アクションはマーカーの個数を返します。

```
markersLength()
```

【showMap】

showMap アクションは緯度・経度で表された地点を地図モジュールに表示します。

```
showMap(data)
```

data 引数には以下のプロパティから成るオブジェクトを与えます。

```
lat     … 表示する地点の緯度
```

```
lng      … 表示する地点の経度
zoom     … 表示倍率
```

【setStyle】

setStyle アクションは、地図モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項では地図モジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Click】

Click イベントは地図モジュールがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティにクリックされた地点に関する情報が渡されます。

```
evt.latlng.lat … クリックされた地点の緯度
evt.latlng.lng … クリックされた地点の経度
```

【Move】

Move イベントは地図モジュールの表示地点が変更されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMove: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに新しい表示地点に関する情報が渡されます。

```
evt.latlng.lat … 新しい表示地点の緯度
evt.latlng.lng … 新しい表示地点の経度
```

このイベントはマウスでのドラッグ中には発生せず、ドラッグ操作が完了した時点で発生します。

【Zoom】

Zoom イベントは地図モジュールのスケールが変更されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onZoom: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。また、evt オブジェクトの以下のプロパティに情報が渡されます。

```
evt.zoom … 倍率
```

【グラフ】

グラフモジュールはデータを可視化するグラフを表示するためのモジュールです。

【アクション】

本項ではグラフモジュールのアクションについて説明します。

【downloadImage】

downloadImage アクションはグラフを PNG 画像ファイルとしてダウンロードします。

```
downloadImage()
```

【setData】

setData アクションはグラフ表示に用いるデータを設定します。

```
setData(data)
```

data 引数には以下のプロパティから成るオブジェクトを与えます。

```
type     … グラフのタイプ。以下のタイプのいずれか:
         BarChart … 積み上げ棒グラフ
         GroupedBarChart … 棒グラフ
         LineChart … 折れ線グラフ
         AreaChart … 面グラフ
         PieChart … 円グラフ
data     … 以下のプロパティから成るデータ系列オブジェクトの配列:
         label    … データ系列のラベル
         values   … データ系列。{x: X値, y: Y値} の形式のオブジェクトの配列
margin   … 余白の幅を指定する、以下のプロパティから成るオブジェクト:
         top      … 上の余白幅
         bottom   … 下の余白幅
         left     … 左の余白幅
         right    … 右の余白幅
```

【setStyle】

setStyle アクションは、グラフモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【キャンバス】

キャンバスモジュールは、スクリプトにより画面上に図形や文字列を描画するモジュールです。

キャンバスモジュールに図形を描画するには、まず【getContext】アクションで 2 次元グラフィックスコンテキストを得ます。次に、コンテキストオブジェクトの各種メソッドを呼び出して図形を描画します。例として、CANVAS1 という名前のキャンバスモジュールに塗りつぶした矩形を青で、矩形の枠線を赤で描画するスクリプトを以下に示します。

```
var ctx = this.items.CANVAS1.getContext();
ctx.fillStyle = "blue";
ctx.fillRect(120, 120, 200, 150);
ctx.strokeStyle = "red";
ctx.strokeRect(100, 100, 200, 150);
```

2 次元グラフィックスコンテキストの描画メソッドの詳細については HTML5 のキャンバス要素に関する解説書やオンラインチュートリアル（例えば下記 URL）などを参照してください。

```
https://developer.mozilla.org/ja/docs/Web/Guide/HTML/Canvas_tutorial/Drawing_shapes
```

【アクション】

本項ではキャンバスモジュールのアクションについて説明します。

【getDOM】

getDOM アクションは、HTML のキャンバス要素を表す DOM ノードを返します。

```
getDOM()
```

【getContext】

getContext アクションは、2 次元グラフィックスコンテキストを返します。

```
getContext()
```

【setCanvasRatio】

setCanvasRatio アクションは、キャンバスの表示倍率を設定します。

```
setCanvasRatio(ratio)
```

ratio 引数にはキャンバス倍率を与えます。

【setStyle】

setStyle アクションは、キャンバスのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではキャンバスモジュールから発生するイベントについて説明します。一部のイベントハンドラの evt 引数に渡されるイベントオブジェクトについては【イベント API】の節を参照してください。

【Blur】

Blur イベントはキャンバスから入力フォーカスが外れたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onBlur: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Click】

Click イベントはキャンバスがクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【ContextMenu】

ContextMenu イベントはコンテキストメニュー（Windows では右クリックメニュー）が開かれたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onContextMenu: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【DoubleClick】

DoubleClick イベントはキャンバスがダブルクリックされたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onDoubleClick: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Focus】

Focus イベントはキャンバスに入力フォーカスが移ってきたときに生じます。イベント

ハンドラは次のように定義します。

```
モジュール名_onFocus: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyDown】

KeyDown イベントはキャンバス上でキーボードのキーが押し下げられたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyPress】

KeyPress イベントはキャンバス上でキーボードのキーが押されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyPress: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【KeyUp】

KeyUp イベントはキャンバス上で押下されたキーボードのキーが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onKeyUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseDown】

MouseDown イベントはキャンバス上でマウスボタンが押下されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseDown: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【MouseUp】

MouseUp イベントはキャンバス上で押されたマウスボタンが離されたときに生じます。イベントハンドラは次のように定義します。

```
モジュール名_onMouseUp: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【Update】

Update イベントはキャンパスモジュールの再描画が必要なときに生じます。イベント

ハンドラは次のように定義します。

```
モジュール名_onUpdate: function(evt){ ... }
```

イベントハンドラの引数 evt にはイベントオブジェクトが渡されます。

【スクリーンコンテナ】

スクリーンコンテナモジュールは、別のスクリーンを画面上に埋め込むためのモジュールです。埋め込まれる側の別スクリーンを「内側」、スクリーンコンテナが設けられたスクリーンを「外側」と呼びます。内側スクリーンの埋め込み方法には次の 2 つがあります。

```
a.  1つのスクリーンコンテナ内に1つだけ内側スクリーンを表示する(デフォルト)。
b.  内側スクリーンを0個以上複製して反復表示する。
```

内側スクリーンの反復表示回数を指定するには後述の setIterate アクションを用います。デフォルトの反復表示回数は 1 です。

【アクション】

本項ではスクリーンコンテナモジュールのアクションについて説明します。

【iterate】

iterate アクションは、内側スクリーンの反復表示の際に個々の内側スクリーンに渡されるデータを設定します。

```
iterate(data)
```

data 引数には反復回数と同じ要素数の配列を与えます。配列の要素についてはデータ構造の規定はなくアプリケーションごとに自由に決められます。与えられたデータは後述の Iterate イベントを介して内側スクリーンへ渡されます。

【screens】

screens アクションは、内側スクリーンのスクリーンオブジェクトを外側スクリーンから参照するための関数です。

```
screens(index)
```

index 引数には参照する内側スクリーンの番号を与えます。たとえばスクリーンコンテナ SCREENCONTAINER1 の中に反復表示された i 番目の内側スクリーンの中のテキストボックス TEXTBOX1 の値を得るには次のようにします。

```
var val = this.items.SCREENCONTAINER1.screens(i).items.TEXTBOX1.getValue();
```

【setIterate】

setIterate アクションは、内側スクリーンの反復表示回数を設定します(デフォルトは 1)。

```
setIterate(iterate);
```

iterate 引数にはスクリーンの反復表示回数を整数値で与えます。

【setScreen】

setScreen アクションは、スクリーンコンテナ内に表示する別スクリーンを変更します。

```
setScreen(screen)
```

screen 引数には新しい内側スクリーンの名前を文字列で与えます。

【setStyle】

setStyle アクションは、スクリーンコンテナモジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【イベント】

本項ではスクリーンコンテナから発生するイベントについて説明します。

【Iterate】

Iterate イベントは、スクリーンコンテナの iterate アクションが実行されたときに発生して内側スクリーンに伝播します。このイベントを内側スクリーンで受け取るにはイベントハンドラを次のように定義します。

```
onIterate: function(data){ ... }
```

イベントハンドラの引数 data には iterate アクションの引数として与えられたデータ配列の要素（i 番目の反復表示スクリーンに対して i 番目の配列要素）が渡されます。

【IFrame】

IFrame モジュールは HTML の <iframe> タグを用いて画面内にネット上のコンテンツ（ウェブページなど）を埋め込むためのモジュールです。

【アクション】

本項では IFrame モジュールのアクションについて説明します。

【setSrc】

setSrc アクションは IFrame モジュール内に表示するコンテンツの URL を設定します。

```
setSrc(src)
```

src 引数には表示する URL を文字列で与えます。

【setStyle】

setStyle アクションは、IFrame モジュールのスタイルを設定します。

```
setStyle(style)
```

style 引数にはスタイルオブジェクトを与えます。

【モデルオブジェクト】

DB テーブル設計で定義されるテーブルおよび DB ビュー設計で定義されるビューをまとめて「モデル」と呼び、モデルの設計情報を表すオブジェクトのことを「モデルオブジェクト」と呼びます。Buddy アプリを構成するモデルは、スクリーンプログラムにおいて this.models で参照できます。たとえば Shohin テーブルの設計情報は次のように参照できます。

```
this.models.Shohin
```

モデルの設計情報は、テーブルの場合は ModelTable クラス、ビューの場合は ModelView クラスで表されます。

【ModelTable】

ModelTableはDBテーブルのモデル情報とレコード操作のためのメソッドを提供するクラスです。

【プロパティ】

【modelType】

modelType プロパティの値は文字列 "table" です。モデルオブジェクトが DB テーブルかどうかを調べるのに用います。

【conf】

confプロパティの値はテーブルの設計情報オブジェクトです。設計情報オブジェクトは以下のプロパティから成ります。

```
TableDbname … DB上のテーブル名(英数_)
TableDisplayname … 表示上のテーブル名(日本語可)
TableType … テーブルタイプ(STOCK または FLOW)
TableFlags … テーブルフラグの配列
TableMemo … 備考(開発上のメモ)
TableColumns … 次のカラム情報オブジェクトの配列
        ColumnDbname … DB上のカラム名(英数_)
        ColumnDisplayname … 表示上のカラム名(日本語可)
        ColumnCategory … データ型のカテゴリ
        ColumnClass … データ型のクラス
        ColumnType … データ型
        ColumnDefaultvalue … デフォルト値
        ColumnOrder … カラムの順序(整数値)
        ColumnFlags … カラムフラグの配列
        ColumnChoices … 選択肢の配列
        ColumnRange … 範囲選択肢(「最小値:最大値:刻み値」)
        ColumnMemo … 備考(開発上のメモ)
TablePrimaryKeyColumns … Primary Key指定するカラム名の配列
TableIndexes … 次のインデックス情報オブジェクトの配列
        IndexDbname … DB上のインデックス名(英数_)
        IndexDisplayname … 表示上のインデックス名(日本語可)
        IndexUnique … Uniqueかどうか(trueまたはfalse)
        IndexColumnsOrExpressions … カラム名または式の配列
TableRules … 次のルール情報オブジェクトの配列
        RuleDbname … DB上の制約名(英数_)
        RuleDisplayname … 表示上のルール名(日本語可)
        RuleCheck … 検査式
        ※特殊な指定として、「カラム名 NOT NULL」「UNIQUE(カラム名,…)」
TableOrders … 次の順序情報オブジェクトの配列
        OrderName … 順序名(英数_)
```

```
        OrderDisplayname … 表示上の順序名(日本語可)
        OrderDefault … デフォルト順序かどうか(trueまたはfalse)
        OrderSpecs … 次の順序カラム指定オブジェクトの配列
                OrderColumn … カラム名
                OrderDirection … ASC または DESC
TableAuth … アクセス制御オブジェクト
```

【name】

name プロパティの値はテーブルの DB 名を表します。この値は conf.TableDbname プロパティと同一です。

【header】

header プロパティの値はテーブルのカラムを表すオブジェクトの配列です。各要素は以下のプロパティから成ります。

```
name     … カラムDB名
displayName … カラム表示名
```

配列の i 番目の要素は conf.TableColumns プロパティの i 番目の要素に対応しています。

【メソッド】

本節では ModelTable クラスのメソッドについて説明します。

【as】

as メソッドは後述の【select】メソッドで作成するサブクエリオブジェクトに別名を与えます。

```
as(name)
```

name 引数にはサブクエリの別名を文字列で与えます。as メソッドは戻り値として ModelTable オブジェクトを返します。この戻り値を使ってさらに select メソッドを呼び出せます。以下に例を示します。

```
var tableName = "weather";
var subquery = this.tables[tableName].as('t').select(["max(t.temp_lo)"], {});
var options = {
    where: {"temp_lo": {op: "=", ref: subquery}},
};
this.tables[tableName].readData(options, (function(error, data) {
    ...
});
```

【crossJoin】

crossJoin メソッドは DB テーブルと別の DB テーブル・ビューまたはサブクエリの交差

結合（cross join）を生成します。

```
crossJoin(table, asname)
```

table 引数には別の DB テーブル･ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。crossJoin メソッドは ModelTable オブジェクトを返します。

【deleteRecord】

deleteRecord メソッドは DB テーブルに格納されたレコードを削除します。

```
deleteRecord(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
where    … 削除するレコードを選択する絞り込み条件。後述の【絞り込み条件】の節を参照してください。
details  … 明細テーブルのレコード削除操作を指定するオブジェクトの配列（後述）。明細テーブルが無ければ省略可
```

details プロパティの配列要素には次のプロパティから成るオブジェクトを与えます。

```
name       … 明細テーブル名
baseColumn … ベーステーブルの参照元カラム名
detailColumn … 明細テーブルの参照先カラム名
returning … 結果オブジェクト（後述）として値を得たいカラム名の配列
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
body     … 本テーブルに対する結果オブジェクト
details  … 明細テーブル（options引数のdetailsプロパティの配列要素）と1対1に対応する要素から成る配列。この配列の要素は、次のプロパティから成るオブジェクトの配列
    body     … 明細テーブルの操作（削除）の結果オブジェクト
    err      … 上記の操作が成功のときはnull、失敗のときはErrorオブジェクト
```

ここで結果オブジェクトは以下のプロパティから成るオブジェクトです。

```
command  … 本テーブル、明細テーブル共に"DELETE"
rowCount … レコード数
rows     … レコードの配列。各レコードは｛カラム名：値，…｝の形式のオブジェクト
```

なお、options 引数に与えるオブジェクトには returning プロパティはありません。ベーステーブルの結果オブジェクトには削除されたレコードのすべてのカラムが含まれます。

【expression】

expression メソッドは後述の【updateRecord】メソッドでカラム値を更新するための SQL 式を表すオブジェクトを生成します。

```
expression(expr, ...)
```

引数には SQL 式を文字列で与えます。複数の引数が与えられたら、すべてを空白でつないで 1 つの SQL 式とみなします。戻り値は SQL 式を表すオブジェクトです。このオブジェクトは updateRecord メソッドの options 引数の data プロパティに与えるカラム値として指定できます。例えば SQL 式を用いてテーブル table1 のカラム num の値を 1 増やすには次のように記述します。

```
var tableName = "table1";
var table = this.models[tableName];
var options = {
    data: {
        num: table.expression('num + 1'),
    },
};
table.updateRecord(options, function (err, res) {
    ...
});
```

【findNext】

findNext メソッドは DB テーブルのレコードについてキーワード検索を実行します。

```
findNext(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
where    … キーワード検索の対象となるレコードを選択する絞り込み条件。後述の【絞り込み
         条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
keyword  … 検索キーワード。後述の【パタン文字列】の節を参照してください。
searchPos … キーワード検索の開始レコード番号（最初のレコードが0）。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。キーワード検索に成功した場合、コールバック関数の第 1 引数には null、第 2 引数には searchPos 以降のレコードの中で検索キーワードが含まれる最初のレコードの番号が渡されます。検索キーワードが見つからなかった場合には -1 が渡されます。

【fullOuterJoin】

fullOuterJoin メソッドは DB テーブルと別の DB テーブル・ビューまたはサブクエリの完全外部結合（full outer join）を生成します。

```
fullOuterJoin(table, asname)
```

table 引数には別の DB テーブル･ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。後述の on メソッドまたは using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。fullOuterJoin メソッドは ModelTable オブジェクトを返します。

【getCount】

getCount メソッドは与えられた絞り込み条件を満たすレコードの数を返します。

```
getCount(options, callback)
```

options 引数にはデータ読み込みのオプションを指定する、以下のプロパティから成るオブジェクトを与えます。

```
where     … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはレコード数が数値で渡されます。

【innerJoin】

innerJoin メソッドは DB テーブルと別の DB テーブル・ビューまたはサブクエリの内部結合（inner join）を生成します。

```
innerJoin(table, asname)
```

table 引数には別の DB テーブル･ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。後述の on メソッドまたは using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。innerJoin メソッドは ModelTable オブジェクトを返します。次の例のように on メソッドや readData メソッドなどを続けて記述できます。

```
this.models['shain']
  .innerJoin('busho').on('shain.busho_ID', 'busho.ID')
  .readData({}, function (err, res) {...})
```

【insertRecord】

insertRecord メソッドは DB テーブルに新しいレコードを追加（挿入）します。

```
insertRecord(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
data     … 新しいレコードを表す {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト
details  … 明細テーブルへのレコード追加(挿入)操作を指定するオブジェクトの配列(後述)。明細テーブルが無ければ省略可
```

details プロパティの配列要素には次のプロパティから成るオブジェクトを与えます。

```
name       … 明細テーブル名
baseColumn … ベーステーブルの参照元カラム名
detailColumn … 明細テーブルの参照先カラム名
data       … 新しいレコードの配列。各レコードは {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト
returning … 結果オブジェクト(後述)として値を得たいカラム名の配列
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
body     … 本テーブルに対する結果オブジェクト
details  … 明細テーブル(options引数のdetailsプロパティの配列要素)と1対1に対応する要素から成る配列。この配列の要素は、次のプロパティから成るオブジェクトの配列
             body    … 明細テーブルの操作(削除および挿入)の結果オブジェクト
             err     … 上記の操作が成功のときはnull、失敗のときはErrorオブジェクト
```

ここで結果オブジェクトは以下のプロパティから成るオブジェクトです。

```
command  … 本テーブルの場合は"INSERT"、明細テーブルの場合は"DELETE"または"INSERT"
rowCount … レコード数
rows     … レコードの配列。各レコードは {カラム名: 値, …} の形式のオブジェクト
```

なお、options 引数に与えるオブジェクトには returning プロパティはありません。ベーステーブルの結果オブジェクトには挿入されたレコードのすべてのカラムが含まれます。

【leftOuterJoin】

leftOuterJoin メソッドは DB テーブルと別の DB テーブル・ビューまたはサブクエリの左外部結合（left outer join）を生成します。

```
leftOuterJoin(table, asname)
```

table 引数には別の DB テーブル・ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。後述の on メソッドまたは using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。leftOuterJoin メ

ソッドは ModelTable オブジェクトを返します。

【on】

on メソッドは結合条件（JOIN ON 節）を指定します。

```
on(leftColumn, rightColumn, options)
```

leftColumn 引数と rightColumn 引数には結合する 2 つの DB テーブル･ビューまたはサブクエリのカラム名を指定します。options には以下のプロパティから成るオブジェクトを与えます。

```
op        … 結合条件の比較演算子（デフォルトは '='）
```

options 引数は省略でき、その場合は比較演算子として = が用いられます。on メソッドは ModelTable オブジェクトを返します。

【outputCrossData】

outputCrossData メソッドは DB テーブルにクロス集計を適用して得られたレコードをファイルに書き出します。

```
outputCrossData(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
crossOptions … クロス集計オプション。後述の【クロス集計オプション】の節を参照してください。
filters  … 出力フィルタ。カラム名とフィルタ名の対を {"カラム名1": フィルタ名1, "カラム名2": フィルタ名2, …} の形式のオブジェクトで与えます。フィルタについては後述の【フィルタAPI】の節を参照してください。
format   … 出力ファイル形式。現在は "xlsx"（Excelファイル）のみサポートしています。
fileName … 出力ファイル名。省略するとDBテーブル名がファイル名として用いられます。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ出力が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成る結果オブジェクトが渡されます。

```
fileName … 出力ファイル名
url      … 出力ファイルのダウンロードURL
```

【outputData】

outputData メソッドは DB テーブルのレコードをファイルに書き出します。

```
outputData(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
columns  … カラムの並び。出力対象のカラム名を文字列配列で与えます。省略するとDBテーブ
         ルのすべてのカラムが出力されます。
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
num      … 読み込み件数。
offset   … 先頭レコード番号（最初のレコードが0）。
filters  … 出力フィルタ。カラム名とフィルタ名の対を {"カラム名1": フィルタ名1, "カラ
         ム名2": フィルタ名2, …} の形式のオブジェクトで与えます。フィルタについては後
         述の【フィルタAPI】の節を参照してください。
format   … 出力ファイル形式。現在は "xlsx"（Excelファイル）のみサポートしています。
fileName … 出力ファイル名。省略するとDBテーブル名がファイル名として用いられます。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ出力が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成る結果オブジェクトが渡されます。

```
fileName … 出力ファイル名
url      … 出力ファイルのダウンロードURL
```

【readCrossData】

readCrossData メソッドは DB テーブルにクロス集計を適用して得られたレコードを返します。

```
readCrossData(crossOptions, callback)
```

crossOptions 引数にはクロス集計オプションを与えます。クロス集計オプションについては後述の【クロス集計オプション】の節を参照してください。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはレコードの配列が渡されます。各レコードは { カラム名 1: 値 1, カラム名 2: 値 2, ...} の形式のオブジェクトです。

【readCrossGraphData】

readCrossGraphData メソッドは DB テーブルにクロス集計を適用して得られたレコードをグラフモジュール向けに整形して返します。

```
readCrossGraphData(options, callback)
```

crossOptions 引数にはクロス集計オプションを与えます。クロス集計オプションについては後述の【クロス集計オプション】の節を参照してください。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）

が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはグラフモジュール向けに整形されたレコードの配列が渡されます。グラフモジュール向けのレコードの構成については【グラフ】の節を参照してください。

【readData】

readData メソッドは DB テーブルからデータ行を読み込みます。

```
readData(options, callback)
```

options 引数にはデータ読み込みのオプションを以下のプロパティから成るオブジェクトで与えます。

```
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
num      … 読み込み件数。
offset   … 先頭レコード番号（最初のレコードが0）。
useCache … キャッシュ利用フラグ。真ならキャッシュを用います。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には得られたレコードの配列が渡されます。各レコードはカラム名と値の対から成る { カラム名 1: 値 1, カラム名 2: 値 2, ...} の形式のオブジェクトで表されます。

【rightOuterJoin】

rightOuterJoin メソッドは DB テーブルと別の DB テーブル・ビューまたはサブクエリの右外部結合（right outer join）を生成します。

```
rightOuterJoin(table, asname)
```

table 引数には別の DB テーブル・ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。前述の on メソッドまたは後述の using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。rightOuterJoin メソッドは ModelTable オブジェクトを返します。

【select】

select メソッドは絞り込み条件および結合に用いられる【サブクエリ】オブジェクトを作成します。

```
select(columns, options)
```

columns 引数にはカラム式の並びを表す配列を与えます。配列の要素はカラム式の文字列または以下のプロパティからなるオブジェクトです。expr にはカラム式（カラム名や

任意の SQL 式)、as には別名を与えます。

```
expr     … カラム式
as       … AS名
```

options 引数には上述の【readData】メソッドの options 引数と同じ構造のオプションオブジェクトを与えます。select メソッドは戻り値としてサブクエリオブジェクトを返します。サブクエリオブジェクトの利用方法については【サブクエリ】の節を参照してください。
※この機能は今のところサーバー機能設定では利用できません。将来利用できるように改善する予定です。

【updateRecord】

updateRecord メソッドは DB テーブルに格納された既存レコードを更新します。

```
updateRecord(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
where    … 更新するレコードを選択する絞り込み条件。後述の【絞り込み条件】の節を参照してください。
data     … 更新されるカラムと新しい値の対を表す {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト。値には数値、文字列などのリテラル値に加えて前述の【expression】メソッドで作成したSQL式を表すオブジェクトが指定できます。
details  … 明細テーブルのレコード更新操作を指定するオブジェクトの配列(後述)。明細テーブルが無ければ省略可
```

details プロパティの配列要素には次のプロパティから成るオブジェクトを与えます。

```
name       … 明細テーブル名
baseColumn … ベーステーブルの参照元カラム名
detailColumn … 明細テーブルの参照先カラム名
data       … 新しいレコードの配列。各レコードは {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト
returning … 結果オブジェクト(後述)として値を得たいカラム名の配列
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト(err.response.text にエラー内容)が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
body     … 本テーブルに対する結果オブジェクト
details  … 明細テーブル(options引数のdetailsプロパティの配列要素)と1対1に対応する要素から成る配列。この配列の要素は、次のプロパティから成るオブジェクトの配列
             body    … 明細テーブルの操作(削除および挿入)の結果オブジェクト
             err     … 上記の操作が成功のときはnull、失敗のときはErrorオブジェクト
```

ここで結果オブジェクトは以下のプロパティから成るオブジェクトです。

```
command  … 本テーブルの場合は"UPDATE"、明細テーブルの場合は"DELETE"または"INSERT"
rowCount … レコード数
rows     … レコードの配列。各レコードは {カラム名: 値, …} の形式のオブジェクト
```

なお、options 引数に与えるオブジェクトには returning プロパティはありません。ベーステーブルの結果オブジェクトには更新されたレコードのすべてのカラムが含まれます。また、ベーステーブルについては options 引数の data プロパティで値が与えられたカラムのみが更新され、data プロパティで指定されていないカラムについては何の操作も行いません。一方、明細テーブルはベーステーブルのレコードに関連付けられたすべてのレコードを削除してから details.data プロパティで与えられたレコードを挿入するという手順で更新されます。そのため details.data プロパティで指定されていないカラムの値は NULL（または当該カラムのデフォルト値）になります。

【upsertRecord】

upsertRecord メソッドはレコードの更新を試みて成功したら処理を終えます。もし更新するレコードが無ければ新規レコードの挿入を行ないます。

```
upsertRecord(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
where    … 更新するレコードを選択する絞り込み条件。後述の【絞り込み条件】の節を参照してください。
data     … 更新・挿入されるレコードのカラムと値の対を表す {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト。
details  … 明細テーブルのレコード更新操作を指定するオブジェクトの配列（後述）。明細テーブルが無ければ省略可
```

details プロパティの配列要素には次のプロパティから成るオブジェクトを与えます。

```
name       … 明細テーブル名
baseColumn … ベーステーブルの参照元カラム名
detailColumn … 明細テーブルの参照先カラム名
data       … 新しいレコードの配列。各レコードは {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト
returning … 結果オブジェクト（後述）として値を得たいカラム名の配列
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
body     … 本テーブルに対する結果オブジェクト
details  … 明細テーブル（options引数のdetailsプロパティの配列要素）と1対1に対応する要素から成る配列。この配列の要素は、次のプロパティから成るオブジェクトの配列
```

```
        body    … 明細テーブルの操作(削除および挿入)の結果オブジェクト
        err     … 上記の操作が成功のときはnull、失敗のときはErrorオブジェクト
```

ここで結果オブジェクトは以下のプロパティから成るオブジェクトです。

```
command  … 本テーブルの場合は"INSERT"または"UPDATE"、明細テーブルの場合は"DELETE"または"INSERT"
rowCount … レコード数
rows     … レコードの配列。各レコードは {カラム名: 値, …} の形式のオブジェクト
```

なお、options 引数に与えるオブジェクトには returning プロパティはありません。ベーステーブルの結果オブジェクトには更新されたレコードのすべてのカラムが含まれます。また、ベーステーブルについては options 引数の data プロパティで値が与えられたカラムについて値が挿入または更新されます。data プロパティで指定されていないカラムについては、挿入の場合は NULL（または当該カラムのデフォルト値）が挿入され、更新の場合は何の操作も行いません。一方、明細テーブルはベーステーブルのレコードに関連付けられたすべてのレコードを削除してから details.data プロパティで与えられたレコードを挿入するという手順で更新されます。そのため details.data プロパティで指定されていないカラムの値は NULL（または当該カラムのデフォルト値）になります。

【using】

using メソッドは結合条件（JOIN USING 節）を指定します。

```
using(column1, column2, ...)
```

引数には結合条件となるカラム名を 1 つ以上与えます。using メソッドは ModelTable オブジェクトを返します。

【データ型】

テーブルカラムの型には以下の 5 つがあります。

```
・数値
・真偽値
・文字列
・日付・時間
・ファイル
```

それぞれの型に属する型の詳細および対応する PostgreSQL のデータ型を以下に示します。

```
数値型の詳細(括弧内は有効な範囲):
  ・数値 … numeric(小数点より上は131072桁まで、小数点より下は16383桁まで)
  ・自動連番 … serial8(1から9223372036854775807まで)
  ・整数 … int8(-9223372036854775808から+9223372036854775807まで)
  ・小数点 … float8(15桁精度、およそ1E-307から1E+308まで)
  ・金額 … money(-92233720368547758.08 から +92233720368547758.07まで)
```

```
真偽値型の詳細:
  • 真偽値 … bool
文字列型の詳細:
  • フリー … text
  • 英数字 … text
  • ひらがな … text
  • 全角カナ … text
  • 半角カナ … text
  • 郵便番号 … text
  • メールアドレス … text
日付・時間型の詳細(括弧内は有効な範囲):
  • 日時 … timestamp with time zone(紀元前4713年から294276年まで)
  • 日付 … date(紀元前4713年から5874897年まで)
  • 時間 … time(00:00:00から24:00:00まで)
  • 時間間隔 … interval(-178000000年から178000000年まで)
ファイル型の詳細:
  • フリー … text
  • PDF … text
  • WORD … text
  • EXCEL … text
  • 画像 … text
```

文字列型のうち、フリーの場合は任意の文字列が有効です。その他の型の詳細は以下の正規表現にマッチする文字列のみが有効です。

```
英数字    … ^[A-Za-z0-9_]*$
ひらがな … ^[ / u3041- / u309f / u30a0 / u30fb / u30fc]*$
全角カナ … ^[ / u30a0- / u30ff / u3099- / u309c]*$
半角カナ … ^[ / uff65- / uff9f]*$
郵便番号 … ^ / d{3}-? / d{4}$
メールアドレス
        … ^([A-Za-z0-9._%+-]+@([A-Za-z0-9]+(-[A-Za-z0-9]+)* / .)+[A-Za-z]{2,})?$
```

一方、JavaScript では整数も実数もすべて国際規格 IEEE 754 に基づく倍精度浮動小数点数で表されます。したがって JavaScript で正しく扱える整数は -9007199254740991 から 9007199254740991 まで(絶対値が 2 の 53 乗未満の数)です。JavaScript で扱える実数の範囲は PostgreSQL の float8 と同じです。

【型キャスト】

文字列以外のデータ型の定数は、文字列と型キャストを組み合わせた以下の形式の SQL 式で記述します。

```
'string'::type
```

'string' には文字列、type には PostgreSQL のデータ型の名前を与えます(前節の【データ型】の項を参照してください)。型キャストの用例を以下に示します。

```
'123'::int8 … 整数
'123.45'::float8 … 小数点
't'::bool … 真偽値
'2016-12-25 12:34:56 +0900'::timestamp with time zone … 日時
'2016-12-25'::date … 日付
'12:34:56'::time … 時間
'1 year 2 months 3 days'::interval … 時間間隔
```

型キャストは SQL 式の結果を別のデータ型に変換するのにも用いられます。以下に例を示します。

```
AGE("birthday")::text … 時間間隔から文字列へ
("date" || ' ' || "time")::timestamp … 文字列から日時へ
NULL::text … 文字列型のNULLへ
```

【ModelView】

ModelView は DB ビューのモデル情報とレコード操作のためのメソッドを提供するクラスです。

【プロパティ】

【modelType】

modelType プロパティの値は文字列 "view" です。モデルオブジェクトが DB ビューかどうかを調べるのに用います。

【conf】

conf プロパティの値はビューの設計情報オブジェクトです。設計情報オブジェクトは以下のプロパティから成ります。

```
ViewType … ビューの型(LIST、SUM、CROSS)
ViewDbname … DB上のビュー名(英数_)
ViewDisplayname … 表示上のビュー名(日本語可)
ViewMemo … 備考(開発上のメモ)
ViewColumns … 次のカラム情報オブジェクトの配列
        ColumnAsname … DB上のAS名(英数_)
        ColumnDisplayname … 表示上のカラム名(日本語可)
        ColumnExpression … カラム内容の式
        ColumnSumType … 集計型(GROUP、SUM、COUNT、MAX、MIN、AVG、NONE)
        ColumnOrder … カラムの順序(整数値)
        ColumnMemo … 備考(開発上のメモ)
ViewFrom … テーブル/ビュー名 または次のオブジェクト(入れ子可)
        FromLeft … 左側のテーブル/ビュー名 またはViewFromオブジェクト
        FromLeftAsname … 左側のテーブル/ビューのAS名
        FromRight … 右側のテーブル/ビュー名 またはViewFromオブジェクト
        FromRightAsname … 右側のテーブル/ビューのAS名
        JoinType … JOIN指定(JOIN、LEFT JOIN、RIGHT JOIN、FULL JOIN)
        JoinOn   … 次のJOIN ON条件式オブジェクトの配列
                ColumnLeft … 左側のテーブル/ビューのカラム名
                ColumnRight … 右側のテーブル/ビューのカラム名
                Operator … 演算子(=、!=)
ViewWhere … ビューのWHERE式
ViewHaving … ビューのHAVING式
ViewOrderBy … 次のビューORDER BY指定オブジェクトの配列
        OrderColumn … カラム名
        OrderDirection … ソート順序(ASC、DESC、NONE)
ViewCross … クロス集計(CROSS)型ビューのオプション
        CrossColumns … 列ラベルカラムのDB名の配列
        CrossRows … 行ラベルカラムのDB名の配列
        CrossValues … 値カラムのDB名の配列
```

```
        CrossFunc … 集計関数(SUM、COUNT、MAX、MIN、AVG)
ViewAuth … アクセス制御オブジェクト
```

【name】

name プロパティの値はビューの DB 名を表します。この値は conf.ViewDbname プロパティと同一です。

【header】

header プロパティの値はビューのカラムを表すオブジェクトの配列です。各要素は以下のプロパティから成ります。

```
name      … カラムDB名
displayName … カラム表示名
```

配列の i 番目の要素は conf.ViewColumns プロパティの i 番目の要素に対応しています。

【メソッド】

【as】

as メソッドは後述の【select】メソッドで作成するサブクエリオブジェクトに別名を与えます。

```
as(name)
```

name 引数にはサブクエリの別名を文字列で与えます。as メソッドは戻り値として ModelView オブジェクトを返します。この戻り値を使ってさらに select メソッドを呼び出せます。

【crossJoin】

crossJoin メソッドは DB ビューと別の DB テーブル・ビューまたはサブクエリの交差結合（cross join）を生成します。

```
crossJoin(table, asname)
```

table 引数には別の DB テーブル･ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。crossJoin メソッドは ModelView オブジェクトを返します。

【findNext】

findNext メソッドは DB ビューのレコードについてキーワード検索を実行します。

```
findNext(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
where     … キーワード検索の対象となるレコードを選択する絞り込み条件。後述の【絞り込み
          条件】の節を参照してください。
order     … 並べ替え。後述の【並べ替え】の節を参照してください。
keyword   … 検索キーワード。後述の【パタン文字列】の節を参照してください。
searchPos … キーワード検索の開始レコード番号（最初のレコードが0）。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。キーワード検索に成功した場合、コールバック関数の第 1 引数には null、第 2 引数には searchPos 以降のレコードの中で検索キーワードが含まれる最初のレコードの番号が渡されます。検索キーワードが見つからなかった場合には -1 が渡されます。

【fullOuterJoin】

fullOuterJoin メソッドは DB ビューと別の DB テーブル・ビューまたはサブクエリの完全外部結合（full outer join）を生成します。

```
fullOuterJoin(table, asname)
```

table 引数には別の DB テーブル・ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。後述の on メソッドまたは using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。fullOuterJoin メソッドは ModelView オブジェクトを返します。

【getCount】

getCount メソッドは与えられた絞り込み条件を満たすレコードの数を返します。

```
getCount(options, callback)
```

options 引数にはデータ読み込みのオプションを指定する、以下のプロパティから成るオブジェクトを与えます。

```
where     … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはレコード数が数値で渡されます。

【innerJoin】

innerJoin メソッドは DB ビューと別の DB テーブル・ビューまたはサブクエリの内部結合（inner join）を生成します。

```
innerJoin(table, asname)
```

table 引数には別の DB テーブル･ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。後述の on メソッドまたは using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。innerJoin メソッドは ModelView オブジェクトを返します。

【leftOuterJoin】

leftOuterJoin メソッドは DB ビューと別の DB テーブル･ビューまたはサブクエリの左外部結合（left outer join）を生成します。

```
leftOuterJoin(table, asname)
```

table 引数には別の DB テーブル･ビュー名、ModelTable オブジェクト、ModelView オブジェクト、または【サブクエリ】オブジェクトを与えます。asname 引数は table 引数に与えた結合対象の別名を指定します。別名は table 引数がサブクエリの場合は必須、それ以外の場合は省略可能です。後述の on メソッドまたは using メソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。leftOuterJoin メソッドは ModelView オブジェクトを返します。

【on】

on メソッドは結合条件（JOIN ON 節）を指定します。

```
on(leftColumn, rightColumn, options)
```

leftColumn 引数と rightColumn 引数には結合する 2 つの DB テーブル･ビューまたはサブクエリのカラム名を指定します。options には以下のプロパティから成るオブジェクトを与えます。

```
op        … 結合条件の比較演算子（デフォルトは '='）
```

options 引数は省略でき、その場合は比較演算子として = が用いられます。on メソッドは ModelView オブジェクトを返します。

【outputCrossData】

outputCrossData メソッドは DB ビューにクロス集計を適用して得られたレコードをファイルに書き出します。

```
outputCrossData(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
crossOptions … クロス集計オプション。後述の【クロス集計オプション】の節を参照してください。
filters  … 出力フィルタ。カラム名とフィルタ名の対を {"カラム名1": フィルタ名1, "カラム名2": フィルタ名2, …} の形式のオブジェクトで与えます。フィルタについては後述の【フィルタAPI】の節を参照してください。
```

```
format   … 出力ファイル形式。現在は "xlsx"(Excelファイル)のみサポートしています。
fileName … 出力ファイル名。省略するとDBテーブル名がファイル名として用いられます。
```

【outputData】

outputData メソッドは DB ビューのレコードをファイルに書き出します。

```
outputData(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
columns  … カラムの並び。出力対象のカラム名を文字列配列で与えます。省略するとDBビュー
           のすべてのカラムが出力されます。
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
num      … 読み込み件数。
offset   … 先頭レコード番号(最初のレコードが0)。
filters  … 出力フィルタ。カラム名とフィルタ名の対を {"カラム名1": フィルタ名1, "カラ
           ム名2": フィルタ名2, …} の形式のオブジェクトで与えます。フィルタについては後
           述の【フィルタAPI】の節を参照してください。
format   … 出力ファイル形式。現在は "xlsx"(Excelファイル)のみサポートしています。
fileName … 出力ファイル名。省略するとDBビュー名がファイル名として用いられます。
```

【readCrossData】

readCrossData メソッドは DB ビューにクロス集計を適用して得られたレコードを返します。

```
readCrossData(crossOptions, callback)
```

crossOptions 引数にはクロス集計オプションを与えます。クロス集計オプションについては後述の【クロス集計オプション】の節を参照してください。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト(err.response.text にエラー内容)が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはレコードの配列が渡されます。各レコードは { カラム名 1: 値 1, カラム名 2: 値 2, ...} の形式のオブジェクトです。

【readCrossGraphData】

readCrossGraphData メソッドは DB ビューにクロス集計を適用して得られたレコードをグラフモジュール向けに整形して返します。

```
readCrossGraphData(options, callback)
```

crossOptions 引数にはクロス集計オプションを与えます。クロス集計オプションについては後述の【クロス集計オプション】の節を参照してください。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) と

すると、実行が失敗したら第1引数にErrorオブジェクト（err.response.textにエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第1引数にはnull、第2引数にはグラフモジュール向けに整形されたレコードの配列が渡されます。グラフモジュール向けのレコードの構成については【グラフ】の節を参照してください。

【readData】

readDataメソッドはDBビューからデータ行を読み込みます。

```
readData(options, callback)
```

options 引数にはデータ読み込みのオプションを以下のプロパティから成るオブジェクトで与えます。

```
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
num      … 読み込み件数。
offset   … 先頭レコード番号（最初のレコードが0）。
useCache … キャッシュ利用フラグ。真ならキャッシュを用います。
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第1引数にはnull、第2引数には得られたレコードの配列が渡されます。各レコードはカラム名と値の対から成る { カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクトで表されます。

【rightOuterJoin】

rightOuterJoinメソッドはDBビューと別のDBテーブル・ビューまたはサブクエリの右外部結合（right outer join）を生成します。

```
rightOuterJoin(table, asname)
```

table引数には別のDBテーブル・ビュー名、ModelTableオブジェクト、ModelViewオブジェクト、または【サブクエリ】オブジェクトを与えます。asname引数はtable引数に与えた結合対象の別名を指定します。別名はtable引数がサブクエリの場合は必須、それ以外の場合は省略可能です。前述のonメソッドまたは後述のusingメソッドで結合条件を指定できます。いずれも指定しない場合は自然結合（natural join）となります。rightOuterJoinメソッドはModelViewオブジェクトを返します。

【select】

selectメソッドは絞り込み条件および結合に用いられる【サブクエリ】オブジェクトを作成します。

```
select(columns, options)
```

columns 引数にはカラム式の並びを表す配列を与えます。配列の要素はカラム式の文字

列または以下のプロパティからなるオブジェクトです。expr にはカラム式（カラム名や任意の SQL 式）、as には別名を与えます。

```
expr      … カラム式
as        … AS名
```

options 引数には上述の【readData】メソッドの options 引数と同じ構造のオプションオブジェクトを与えます。select メソッドは戻り値としてサブクエリオブジェクトを返します。サブクエリオブジェクトの利用方法については【サブクエリ】の節を参照してください。

※この機能は今のところサーバー機能設定では利用できません。将来利用できるように改善する予定です。

【カラム式】

カラム内容の式には SQL の式を自由に記述できます。本節では、ビューエディタのカラムの式編集ダイアログで選択して入力できる SQL 式について説明します。

【ABS（絶対値）】

ABS 関数は、与えられた値の絶対値を返します。

```
ABS(number)
```

引数 number には値が数値型となる式（たとえば数値リテラルや数値型のカラム名）を与えます。戻り値のデータ型は入力値と同じです。たとえば ABS(-34.5) は 34.5 を返します。

【AGE（日または日時の差）】

AGE 関数は、日付や日時の差を求めます。

```
AGE(timestamp1, timestamp2)
AGE(timestamp1)
```

引数 timestamp1 および timestamp2 には値が日付型または日時型となる式（たとえば日付リテラルや日時型のカラム名）を与えます。戻り値のデータ型は時間間隔型です。引数が 2 つの場合、2 つの日付または日時の差を返します。たとえば AGE('2017-06-15'::timestamp, '1980-01-01'::timestamp) は 37 years 5 mons 14 days を返します。引数が 1 つの場合、現在の日付と与えられた値との差を返します。たとえば現在の日付が 2017 年 6 月 20 日のとき、AGE('1 Jan 2017'::date) は 5 mons 19 days を返します。これは AGE(CURRENT_DATE, '1 Jan 2017'::date) と書くのと同義です。

【ARRAY_AGG（該当レコードの値を集約して配列化）】

ARRAY_AGG 関数は、与えられたレコードの値を集約して配列にします。

```
ARRAY_AGG(column)
```

column 引数には複数の値を返す式（たとえばカラム名）を与えます。戻り値は入力値と同じデータ型の配列です。

【ARRAY_TO_STRING（配列から文字列に変換）】

ARRAY_TO_STRING 関数は配列を文字列に変換します。

```
ARRAY_TO_STRING(array, text1)
ARRAY_TO_STRING(array, text1, text2)
```

array 引数には値が配列の式（たとえば配列リテラルや配列型のカラム名）、text1 引数には区切り文字列を与えます。text2 引数は省略可能で、与えられたときは配列中の NULL 値を表す文字列として用いられます。たとえば ARRAY_TO_STRING(ARRAY[-2, 1, NULL, 1, 2], ',', '*') は -2,-1,*,1,2 を返します。

【CASE（条件式）】

CASE 式は、条件の真偽に応じて別の値を返します。

```
CASE
WHEN <condition> THEN <value1>
ELSE <value2>
END
```

condition には条件式を与えます。また、value1 と value2 には条件が真のときと偽のときに返される値をそれぞれ指定します。条件式は WHEN … THEN … の部分は 2 つ以上繰り返し記述することで複数指定できます。このとき value2 はすべての条件式が偽のときに返される値となります。ELSE … の部分は省略できます。ELSE 句がなく、どの条件も真でないとき、CASE 式の結果は NULL です。

【COALESCE（NULL でない最初の値）】

COALESCE 関数は、与えられた引数のうち最初の NULL でない値を返します。すべての引数が NULL のときは結果は NULL です。

```
COALESCE(value1, value2, ...)
```

以下に用例を示します。

```
COALESCE(display_name, user_id, '<unknown>')
```

この式は display_name が NULL でなければをその値を返します。そうでない場合（値が NULL のとき）は、user_id が NULL でなければその値を返します。さもなければ <unknown> が返されます。

【CONCAT（文字列結合）】

CONCAT 関数は、すべての引数を文字列化して結合した結果を返します。値が NULL の引数は無視されます。

```
CONCAT(value1, value2, ...)
```

引数には任意のデータ型の式（たとえば文字列リテラルやカラム名）を与えます。戻り値のデータ型は文字列型です。

【DATE_PART（年切出）】

この式は DATE_PART 関数を用いて日時型の値から年を取り出します。

```
DATE_PART('year', timestamp)
```

timestamp 引数には値が日付型の式（たとえば日時型のカラム名）を与えます。第 1 引数に以下のフィールド名を与えることで日時の成分を取り出せます。

```
century  … 世紀
day      … 日
decade   … 年を10で割った値
dow      … 日曜日(0)から土曜日(6)までの曜日
doy      … 年内の通算日数(1～366)
epoch    … 1970年1月1日からの経過秒数
hour     … 時
isodow   … 月曜日(1)から日曜日(7)までの曜日
isoyear  … ISO 8601週番号年
microseconds … マイクロ秒(端数を含む秒に1,000,000を乗じた値)
millennium … ミレニアム(1千年期間)
milliseconds … ミリ秒(端数を含む秒に1,000を乗じた値)
minute   … 分
month    … 月
quarter  … 四半期(1～4)
second   … 秒
timezone … UTCからの時間帯オフセット(秒単位)
timezone_hour … 時間帯オフセットの時の成分
timezone_minute … 時間帯オフセットの分の成分
week     … ISO 8601週番号
year     … 年
```

日時フィールド名の詳細については PostgreSQL マニュアルの DATE_PART 関数の節を参照してください。

```
https://www.postgresql.jp/document/9.4/html/functions-datetime.html
```

【FLOOR（切り捨て）】

FLOOR 関数は、与えられた引数より大きくない最大の整数を返します。

```
FLOOR(number)
```

引数 number には値が数値型となる式（たとえば数値リテラルや数値型のカラム名）を与えます。戻り値のデータ型は入力値と同じです。たとえば FLOOR(-45.6) は -46 を返します。

【FORMAT（書式設定された文字列）】

FORMAT 関数は、C 言語の sprintf 関数と同様の方法で整形された文字列を生成します。

```
FORMAT(format, value, ...)
```

format 引数には第 2 引数以降の値の出力書式を決めるフォーマット文字列（フォーマット指定子を含む文字列）を与えます。戻り値のデータ型は文字列型です。フォーマット指定子は % で始まる以下の形式の文字列です。

```
%[position][flags][width]type
```

position は n$ の形式の文字列で、n は出力する引数の番号です（format 引数のあとの最初の引数が 1）。flags に - を指定すると左詰めになります。width には出力する最小の文字幅を与えます。width に * を指定すると次の引数が文字幅として用いられ、*n$ を指定すると n 番目の引数が文字幅として用いられます。
position、flags、width は省略できます。type にはフォーマット変換の形を指定します。以下の型が利用できます。

```
s       … 文字列に変換して出力します。NULL値は空文字列になります。
I       … 必要なら二重引用符で囲まれたSQL識別子に変換して出力します。NULL値はエラーになります。
L       … 引用符で囲まれたリテラルに変換して出力します。NULL値は引用符なしでNULLと出力されます。
```

また、フォーマット指定子の特別な場合として、%% は文字 % を出力します。FORMAT 関数の用例を以下に示します。

```
FORMAT('%s, %s', 'foo', 'bar')
    結果:foo, bar
FORMAT('%s-%s-%s', 1, 2, 3)
    結果:1-2-3
FORMAT('%3$s-%2$s-%1$s', 1, 2, 3)
    結果:3-2-1
FORMAT('%s%%', 12.3)
    結果:12.3%
FORMAT('|%5s|', 'foo')
    結果:|  foo|
FORMAT('|%-5s|', 'foo')
    結果:|foo  |
FORMAT('|%*s|', 5, 'foo')
    結果:|  foo|
FORMAT('|%*2$s|', 'foo', 5, 'bar')
    結果:|  bar|
FORMAT('|%1$*2$s|', 'foo', 5, 'bar')
    結果:|  foo|
format('%I', 'foo');
    結果:foo
```

```
format('%I', '123');
    結果:"123"
format('%L', 'foo');
    結果:'foo'
format('%L', NULL);
    結果:NULL
```

フォーマット指定子の詳細については PostgreSQL マニュアルの format 関数の節を参照してください。

```
https://www.postgresql.jp/document/9.4/html/functions-string.html
```

【INTERVAL（一日後）】

この SQL 式は、与えられた日付の一日後の日付を返します。

```
<date> + '1 day'::INTERVAL
```

<date> の部分には値が日付型の式（たとえば日付型のカラム名）を与えます。この SQL 式のデータ型は <date> の式と同じです。また、'1 day'::INTERVAL の部分には別の時間間隔型リテラルを指定できます。いくつかの例とその意味を以下に示します。

```
'1 year 2 mons 3 days 04:05:06'::INTERVAL … 1年2ヶ月3日4時間5分6秒
'1 year 2 months 3 days 4 hours 5 minutes 6 seconds'::INTERVAL … 同上
'2 weeks'::INTERVAL … 14日
'1.5 year'::INTERVAL … 1年6ヶ月
'1.5 month'::INTERVAL … 1ヶ月15日(1ヶ月＝30日で換算されます)
'1.5 week'::INTERVAL … 10日12時間
'1.5 day'::INTERVAL … 1日12時間
'1.5 hour'::INTERVAL … 1時間30分
'1.5 minute'::INTERVAL … 1分30秒
```

時間間隔型リテラルの詳細については PostgreSQL マニュアルの時間間隔入力の節を参照してください。

```
https://www.postgresql.jp/document/9.4/html/datatype-datetime.html
```

【LEFT（先頭の文字列）】

LEFT 関数は、文字列の先頭（左）から指定文字数分の部分文字列を返します。

```
LEFT(text, number)
```

text 引数には値が文字列型の式（たとえば文字列リテラルや文字列型のカラム名）を与えます。number 引数には文字数を指定します。文字数が負の値のときは、その値の絶対値に等しい文字数の分だけ文字列の末尾を除いた部分文字列を返します。

【LOWER（小文字変換）】

LOWER 関数は、与えられた文字列を小文字に変換します。

```
LOWER(text)
```

text 引数には値が文字列型の式（たとえば文字列型のカラム名）を与えます。戻り値のデータ型は文字列型です。

【NULLIF（条件下で NULL）】

NULLIF 関数は、与えられた 2 つの値が等しければ NULL を返します。

```
NULLIF(value1, value2)
```

引数 value1 と value2 には任意のデータ型の式（たとえばカラム名やリテラル値）を与えます。2 つの値が等しければ結果は NULL です。そうでなければ value1 が返されます。このとき、戻り値のデータ型は value1 と同じです。以下に NULLIF 関数を用いてゼロ除算を避ける例を示します。

```
column1 / NULLIF(column2, 0)
```

この式の結果は、column2 が 0 のときは NULL、そうでなければ column1/column2 です。

```
COALESCE(column1 / NULLIF(column2, 0), 0)
```

この例では上述の COALESCE 関数を併用して column2 が 0 のときの結果を 0 としています。

【RIGHT（末尾の文字列）】

RIGHT 関数は、文字列の末尾（右）から指定文字数分の部分文字列を返します。

```
RIGHT( <text>, <number>)
```

text 引数には値が文字列型の式（たとえば文字列リテラルや文字列型のカラム名）を与えます。number 引数には文字数を指定します。文字数が負の値のときは、その値の絶対値に等しい文字数の分だけ文字列の先頭を除いた部分文字列を返します。

【ROUND（四捨五入）】

ROUND 関数は、与えられた数値を四捨五入して返します。

```
ROUND(number)
```

引数には値が数値型の式（たとえば数値リテラルや数値型のカラム名）を与えます。戻り値のデータ型は入力値と同じです。たとえば ROUND(23.4) は 23 を返します。

【SUBSTR（文字列切出）】

SUBSTR 関数は、与えられた文字列の部分文字列を返します。

```
SUBSTR(text, number1, number2)
```

text 引数には値が文字列型の式（たとえば文字列リテラルや文字列型のカラム名）を与えます。number1 引数には部分文字列の開始位置（文字列の先頭が 0）、number2 引数には部分文字列の文字数を整数で与えます。

【TO_CHAR（文字列化）】

TO_CHAR 関数は、種々のデータ型の値を整形された文字列に変換します。

```
TO_CHAR(value, text)
```

value 引数には整形される値を与えます。text 引数には出力書式を定義するテンプレートを文字列で与えます。TO_CHAR 関数の用例を以下に示します。

```
TO_CHAR('Dec 25 2016'::date, 'YYYY-MM-DD')
    結果:'2016-12-25'
TO_CHAR('2016-12-25'::date, 'Dy, Mon DD YYYY')
    結果:'Sun, Dec 25 2016'
TO_CHAR('12 hours 34 minutes 56 seconds'::interval, 'HH24:MI:SS');
    結果:'12:34:56'
TO_CHAR(123, '9999')
    結果:'  123'
TO_CHAR(123, '0000')
    結果:' 0123'
TO_CHAR(-123, '9999')
    結果:' -123'
TO_CHAR(-123, '0000')
    結果:'-0123'
TO_CHAR(-123, 'FM99999')
    結果:'-123'
TO_CHAR(123, '9999.9')
    結果:'  123.0'
TO_CHAR(123, 'FM9999.9')
    結果:'123.'
TO_CHAR(12.3, '999.999')
    結果:'  12.300'
TO_CHAR(12.3, 'FM999.999')
    結果:'12.3'
TO_CHAR(1234, '99')
    結果:' ##'
TO_CHAR(-1234, '99')
    結果:'-##'
```

書式テンプレートの詳細については PostgreSQL マニュアルのデータ型書式設定関数の節を参照してください。

```
https://www.postgresql.jp/document/9.4/html/functions-formatting.html
```

【UPPER（大文字変換）】

UPPER 関数は、与えられた文字列を大文字に変換します。

```
UPPER(text)
```

text 引数には値が文字列型の式（たとえば文字列型のカラム名）を与えます。戻り値のデータ型は文字列型です。

【api オブジェクト】

api オブジェクトは、スクリーン設計の JavaScript プログラム開発に用いられる各種 API モジュールへのアクセスを提供するオブジェクトです。api オブジェクトの構成要素は以下のとおりです。

- api.constants
- api.request
- api.dbrequest
- api.cookie
- api.filter
- api.outputDialog
- api.dialog
- api.store
- api.KeyValueStore
- api.serverFunction
- api.TextSearch
- api.lib.async
- api.lib.objectAssign
- api.lib.superagent
- api.lib.EventEmitter
- api.lib.Decimal
- api.lib.xml2js

【api.constants】

api.constants モジュールはアプリケーション名や運用時の URL などアプリケーション固有の情報を提供するモジュールです。このモジュールが提供する情報はすべてアプリケーションのビルド時に決まる固定値（定数）です。

【プロパティ】

api.constants モジュールには以下のプロパティがあります。

```
apiVersion … APIのバージョン番号文字列。現在は "1.0" のみ
AppDisplayname … アプリケーションの表示名
appName  … アプリケーション名
conf     … サーバ設定。以下のプロパティから成るオブジェクト
         port      … サーバのポート番号
         host      … サーバのホスト名
         companyName … 契約名（会社名など）【TBC】
fileRoot … ルートURL中のファイル名のルートディレクトリ【TBC】
first_screen … 最初のスクリーン
target   … ビルド対象（"debug" または "release"）
urlRoot  … アプリケーション運用時のルートURL
```

【api.request】

api.request モジュールは Buddy サーバに対して種々の処理要求を送るための API 関数を提供するモジュールです。

【関数】

【customLog】

customLog 関数はカスタムログファイルに文字列を書き込みます。

```
api.request.customLog(text, callback)
```

引数：

```
text     … カスタムログファイルに書き込む文字列
callback … コールバック関数
```

text 引数に与えた文字列は、関数呼び出し時のタイムスタンプ、実行ユーザの名前と共に日付別のカスタムログファイルに書き込まれます。カスタムログファイルは開発画面のログ閲覧機能により閲覧できます。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【deleteFile】

deleteFile 関数はアプリケーション用のファイルシステム上のファイルを削除します。

```
api.request.deleteFile(appName, dir, file, callback)
```

引数：

```
appName  … アプリケーション名
dir      … ディレクトリ名
file     … ファイル名
callback … コールバック関数
```

dir 引数に与えるディレクトリ名は appName 引数で与えるアプリケーションのファイルシステム上の絶対パスです。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【getFileList】

getFileList 関数はアプリケーション用のファイルシステム上のファイルの一覧を得ます。

```
api.request.getFileList(appName, dir, callback)
appName  … アプリケーション名
dir      … ディレクトリ名
callback … コールバック関数
```

dir 引数に与えるディレクトリ名は appName 引数で与えるアプリケーションのファイルシステム上の絶対パスです。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null、第 2 引数に dir ディレクトリ内の各ファイルおよびディレクトリの情報を表すオブジェクトの配列が返されます。配列の要素は次のプロパティから成るオブジェクトです。

```
name     … ファイル名またはディレクトリ名
path     … 当該ファイルまたはディレクトリを指す完全なパス名
dir      … ディレクトリなら真、ファイルなら偽
```

【getHoliday】

getHoliday 関数は 1 年分の祝日の一覧を得ます。

```
api.request.getHoliday(year, callback)
```

引数：

```
year     … 年(西暦)
callback … コールバック関数
```

year 引数には年を整数値で与えます。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトの配列が渡されます。

```
year     … 年(西暦)
month    … 月
day      … 日
name     … 祝日の名前
```

【getLatLngAddress】

getLatLngAddress 関数は住所から緯度・経度で表された位置情報を得ます。

```
api.request.getLatLngAddress(address, options, callback)
```

引数：

```
address  … 住所
options  … オプション
callback … コールバック関数
```

address 引数には検索する住所を文字列で与えます。options 引数は今のところ無視されます。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
lat      … 緯度
lng      … 経度
```

【getUserInfo】

getUserInfo 関数はユーザ情報を得ます。

```
api.request.getUserInfo(callback)
```

引数：

```
callback … コールバック関数
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
user_id  … ユーザID
name     … ユーザ名
group    … 権限グループ名
user_group … 所属するグループ名の配列
guest    … ゲストユーザなら真
```

権限グループ名は以下のいずれかです。

```
admin    … 運用管理・開発者
operator … 運用管理者
developer … 開発者
user     … 一般
trial    … お試し
support  … サポート
```

【getZipAddress】

getZipAddress 関数は郵便番号から住所を検索します。

```
api.request.getZipAddress(zip, options, callback)
```

引数：

```
zip      … 郵便番号
options  … 検索オプション。
```

```
callback … コールバック関数
```

zip 引数に与える郵便番号が % で始まるときは前方一致検索、% で終わる時は後方一致検索を行ないます。options 引数には以下のプロパティから成るオブジェクトを与えます。

```
mode     … 検索モード。'SIMPLE' または 'FULL' のどちらか。
offset   … 先頭レコード番号(最初のレコードが0)。デフォルト値は0。
num      … 読み込み件数。デフォルト値は20。
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数には options.mode の値に応じて以下のプロパティから成るオブジェクトが渡されます。

```
(a) options.modeが'SIMPLE'の場合
  address  … 住所
  kana     … 住所のフリガナ
  zip      … 郵便番号(7桁)
(b) options.modeが'FULL'の場合
  build    … ビル名
  build_kana … ビル名のフリガナ
  city     … regionから市名を抜き出したもの
  city_kana … 市名のフリガナ
  district … regionから郡名を抜き出したもの
  district_kana … 郡名のフリガナ
  floor    … ビルの階
  has_chome … 丁目を有する町域なら真、そうでなければ偽
  has_koaza_banchi … 小字毎に番地が起番されている町域なら真、そうでなければ偽
  has_subtown … 小字、丁目、番地,号があれば真、そうでなければ偽
  is_multi_town … 1つの郵便番号が2つ以上の町域を表す場合は真、そうでなければ偽
  is_multi_zip … 1つの町域が2つ以上の郵便番号で表される場合は真、そうでなければ偽
  old_zip  … 旧郵便番号(5桁)
  pref     … 都道府県名
  pref_kana … 都道府県名のフリガナ
  region   … 市区町村名
  region_id … 全国地方公共団体コード(JIS X0401、X0402)を表す数値
  region_kana … 市区町村名のフリガナ
  subtown  … 小字、丁目、番地,号等
  subtown_kana … 小字、丁目、番地,号等のフリガナ
  town     … 町域名
  town_kana … 町域名のフリガナ
  update_reason … 更新の表示。0なら変更なし、1なら変更あり、2なら廃止
  update_status … 変更理由。0は変更なし、1は市政・区政・町政・分区・政令指定都市施行、2は
          住居表示の実施、3は区画整理、4は郵便区調整等、5は訂正、6は廃止
  ward     … regionから区名を抜き出したもの
```

```
ward_kana … 区名のフリガナ
zip       … 郵便番号(7桁)
```

【keyValueStore】

keyValueStore 関数はアプリケーション固有のキーバリューストアに対する操作を実行します。

```
api.request.keyValueStore(method, key, value, callback)
```

引数：

```
method   … キーバリューストアに対する操作。"set"、"get"のいずれか
key      … キー
value    … 値
callback … コールバック関数
```

method 引数が "set" のときは key 引数で与えるキーの値を value で与える値に更新します。method 引数が "get" のときは key 引数で与えるキーの値を読み出します(value 引数は無視されます)。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数には読み出した値が渡されます。

【logout】

logout 関数はログアウト処理を実行します。

```
api.request.logout(callback)
```

引数：

```
callback … コールバック関数
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【makeDirectory】

makeDirectory 関数はアプリケーション用のファイルシステム上にディレクトリを作成します。

```
api.request.makeDirectory(appName, dir, callback)
```

引数：

```
appName  … アプリケーション名
dir      … 作成するディレクトリ名
```

```
callback … コールバック関数
```

dir 引数に与えるディレクトリ名は appName 引数で与えるアプリケーションのファイルシステム上の絶対パスです。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【outputReport】

outputReport 関数はレポート出力を実行します。

```
api.request.outputReport(appName, outputName, target, param, callback)
```

引数：

```
appName  … アプリケーション名
outputName … レポート名
target   … レポート出力の対象。"debug"または"release"
param    … レポートの制御パラメタ
callback … コールバック関数
```

param 引数には以下のプロパティから成るオブジェクトを与えます。

```
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
offset   … 先頭レコード番号（最初のレコードが0）。
num      … 出力件数。
これら以外のプロパティも指定可で、レポート中で「this.param.プロパティ名」で参照できる。
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null、第 2 引数にはレポート出力プロセスの ID が渡されます。

【renameFile】

renameFile 関数はアプリケーション用のファイルシステム上のファイルの名前を変更します。

```
api.request.renameFile(appName, dir, src, dst, callback)
```

引数：

```
appName  … アプリケーション名
dir      … ディレクトリ名
src      … 元のファイル名
dst      … 新しいファイル名
callback … コールバック関数
```

dir 引数に与えるディレクトリ名は appName 引数で与えるアプリケーションのファイルシステム上の絶対パスです。実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【resizeImage】

resizeImage 関数は画像の縦横サイズを変換します。

```
api.request.resizeImage(options, callback)
```

引数：

```
options  … オプション
callback … コールバック関数
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
format   … 画像フォーマット(gif, jpg, png等)
src      … 変換元ファイル名
dst      … 変換先ファイル名
width    … 横ピクセル数
height   … 縦ピクセル数
ignoreAspectRatio … 真なら元の縦横比を無視する
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【sendMail】

sendMail 関数はサーバを通じてメールを送信します。

```
api.request.sendMail(maildata, callback)
```

引数：

```
maildata … メールデータ(後述)
callback … コールバック関数
```

maildata 引数には以下のプロパティから成るオブジェクトを与えます。

```
type     … メール本文の形式。"text"または"html"
from     … 送信元メールアドレス(省略するとBuddyサーバの設定ファイルで指定された送信
           元メールアドレスを使用)
to       … 宛先メールアドレス(カンマ区切りで複数指定可)
cc       … CCメールアドレス(カンマ区切りで複数指定可)
bcc      … BCCメールアドレス(カンマ区切りで複数指定可)
subject  … 表題
```

```
body        … 本文
attachments … 添付ファイルパス(filesディレクトリからの相対パス)の配列
smtpserver … SMTPサーバのホスト名またはIPアドレス(省略するとBuddyサーバの設定ファイ
        ルで指定されたSMTPサーバを使用)
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、メールの送信に失敗した場合にはコールバック関数の第 1 引数に Error オブジェクト（err.response.text にエラー内容）が渡されます。メールの送信に成功した場合は、第 1 引数には null、第 2 引数には以下のプロパティから成る SMTP 応答オブジェクトが渡されます。

```
envelope … エンベロープオブジェクト
accepted … SMTPサーバによって受理されたメールアドレスの配列
rejected … SMTPサーバによって拒否されたメールアドレスの配列
response … SMTPサーバからの応答メッセージ文字列
```

少なくとも 1 つのメールアドレスが SMTP サーバにより受理されたら送信成功として扱われます。

【sendMessage】

sendMessage 関数はサーバを通じてメッセージを送信します。

```
api.request.sendMessage(messagedata, callback)
```

引数：

```
messagedata … メッセージデータ(後述)
callback … コールバック関数
```

messagedata 引数には以下のプロパティから成るオブジェクトを与えます。

```
type     … メッセージの形式。文字列 "long_message" を与えます。
to       … 宛先。1つ以上の宛先をカンマ区切りの文字列で与えます。各宛先は以下のいずれか
        です。
        • ユーザーID … 個別ユーザー
        • :ALL … 全ユーザー
        • :ONLINE … 全オンラインユーザー
        • :GROUP(グループ名)  … グループに属する全ユーザー
title    … 表題。文字列で与えます。
body     … 本文。文字列で与えます。
option   … オプション。以下のプロパティから成るオブジェクトを与えます。
        offlineuser … 文字列 "sendmail" を与えると、相手がオフラインの場合はメールで
                送信します。
```

実行結果はコールバック関数を通じて返されます。コールバック関数を callback(err, res) とすると、メッセージの送信に失敗した場合にはコールバック関数の第 1 引数に Error オブジェクト（err.response.text にエラー内容）が渡されます。メッセージの送信に成功した場合は、コールバック関数の第 1 引数には null、第 2 引数には各宛先への送信結

果の配列が渡されます。送信結果は以下のいずれかです。

```
・オンラインで送信した場合 … "ユーザーID:socket"
・メールで送信した場合 … "ユーザーID:mailto:メールアドレス"
・メール送信に失敗した場合 … "ユーザーID:sendmail error:エラー内容"
・相手がオフラインで送信しなかった場合 … "ユーザーID:noop"
```

【upload】

upload関数はアプリケーション用のファイルシステム上にファイルをアップロードします。

```
api.request.upload(appName, dir, files, callback)
```

引数：

```
appName  … アプリケーション名
dir      … ディレクトリ名
files    … ファイルリスト
callback … コールバック関数
```

dir引数に与えるディレクトリ名はappName引数で与えるアプリケーションのファイルシステム上の絶対パスです。files引数にはHTML5のFileListオブジェクトを与えます（※注：Internet Explorer 11 では FileList オブジェクトは利用できません）。実行結果はコールバック関数を通じて返されます。コールバック関数をcallback(err, res)とすると、実行が失敗したら第1引数にErrorオブジェクト（err.response.textにエラー内容）が返されます。実行が成功したら第1引数にはnull、第2引数にはアップロードされたフェイルのパス名が渡されます。

【api.dbrequest】

api.dbrequest モジュールは Buddy サーバに対してデータベース処理要求を送るための API 関数を提供するモジュールです。

【関数】

【addListener】

addListener 関数は DB テーブルのテーブルトリガに関するメッセージを受け取るリスナー関数を追加します。

```
api.dbrequest.addListener(appName, tableName, listener, callback)
```

引数：

```
appName  … アプリケーション名
tableName … DBテーブル名
listener … リスナー関数
callback … コールバック関数
```

listener引数にはリスナーとなる関数オブジェクトを与えます。リスナー関数の第1引数にはテーブルトリガに関するメッセージが渡されます。メッセージは以下のプロパティから成るオブジェクトです。

```
type     … "dbi"
api      … "notify"
payload  … メッセージの本体(ペイロード)。以下のプロパティから成るオブジェクト
         NEW       … 新レコード
         OLD       … 旧レコード
         TG_NAME   … トリガ名
         TG_WHEN   … トリガ実行タイミング。"BEFORE"(データ操作実行前)、"AFTER"(データ操
                     作実行後)のいずれか。
         TG_LEVEL … トリガ実行レベル。"ROW"(行単位)、"STATEMENT"(文単位)のいずれか
         TG_OP     … データ操作名。"INSERT"、"UPDATE"、"DELETE"、"TRUNCATE"のいずれか
         TG_RELID … DBテーブルのOID
         TG_TABLE_NAME … DBテーブル名
         TG_TABLE_SCHEMA … スキーマ名
from     … メッセージの送信元
time     … タイムスタンプ
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト(err.response.text にエラー内容)が返されます。実行が成功したら第1引数には null が渡されます。第 2 引数は今のところ使用されません。

【clearCache】

clearCache 関数は後述の readCachedData 関数で用いるキャッシュの内容を破棄します。

```
api.dbrequest.clearCache()
```

【deleteRecord】

deleteRecord 関数は DB テーブルに格納されたレコードを削除します。

```
api.dbrequest.deleteRecord(appName, target, tableName, where, callback)
```

引数：

```
appName   … アプリケーション名
target    … ビルド対象("debug"または"release")
tableName … DBテーブル名
where     … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
callback  … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第1引数にはnull、第2引数には以下のプロパティから成るオブジェクトが渡されます。

```
body      … 削除されたレコードの配列。各レコードは {カラム名1: 値1，カラム名2: 値2，
            ...} の形式のオブジェクト。
```

【findNext】

findNext 関数は DB テーブル・ビューのレコードについてキーワード検索を実行します。

```
api.dbrequest.findNext(appName, target, tableName, where, order, keyword, searchPos,
 callback)
```

引数：

```
appName   … アプリケーション名
target    … ビルド対象("debug"または"release")
tableName … DBテーブル・ビュー名
where     … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order     … 並べ替え。後述の【並べ替え】の節を参照してください。
keyword   … 検索キーワード。後述の【パタン文字列】の節を参照してください。
searchPos … キーワード検索の開始レコード番号（最初のレコードが0）。
callback  … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。キーワード検索に成功した場合、コール

バック関数の第 1 引数には null、第 2 引数には searchPos 以降のレコードの中で検索キーワードが含まれる最初のレコードの番号が渡されます。検索キーワードが見つからなかった場合には -1 が渡されます。

【getColumns】

getColumns 関数は DB テーブル・ビューのカラムに関する情報を得ます。

```
api.dbrequest.getColumns(appName, target, tableName, callback)
```

引数：

```
appName   … アプリケーション名
target    … ビルド対象("debug"または"release")
tableName … DBテーブル・ビュー名
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトの配列が渡されます。

```
a.  DBテーブルの場合
      ColumnDbname … カラム名
      ColumnDisplayname … カラム表示名
      ColumnCategory … データ型のカテゴリ
      ColumnClass … データ型のクラス
      ColumnType … データ型
      ColumnDefaultvalue … デフォルト値
      ColumnOrder … カラムの順序
      ColumnFlags … カラムフラグの配列
      ColumnChoices … 選択肢の配列
      ColumnRange … 範囲選択肢
      ColumnMemo … メモ
b.  DBビューの場合
      ColumnAsname … カラムAS名
      ColumnDisplayname … カラム表示名
      ColumnExpression … カラム内容の式
      ColumnSumType … 集計型（GROUP、SUM、COUNT、MAX、MIN、AVG、NONE）
      ColumnOrder … カラムの順序
      ColumnMemo … メモ
```

【getCount】

getCount 関数は与えられた絞り込み条件を満たすレコードの数を返します。

```
api.dbrequest.getCount(appName, target, tableName, where, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
tableName … DBテーブル･ビュー名
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはレコード数が数値で渡されます。

【getCrossColumns】

getCrossColumns 関数は DB テーブル･ビューのクロス集計結果（一時テーブル）を構成するカラムの情報を得ます。

```
api.dbrequest.getCrossColumns(appName, target, crossOptions, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
crossOptions … クロス集計オプション。後述の【クロス集計オプション】の節を参照してくださ
        い。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトの配列が渡されます。

```
ColumnAsname … カラム名
ColumnDisplayname … カラム表示名
ColumnExpression … カラム内容の式
ColumnValue … 値カラムのカラム名(列ラベルの場合のみ)
ColumnSumType … 集計タイプ。"GROUP"(行ラベルの場合)、または"SUM"、"COUNT"、"MAX"、"MIN"、
        "AVG"のいずれか(列ラベルの場合)
ColumnCategory … カテゴリ階層(列ラベルの場合のみ)。以下のプロパティから成るオブジェク
        トの配列
         CatExpression … ひとつのカテゴリ階層を定義するSQL条件式
         CatValues … 列カラムの値(上記のSQL条件式に現れるリテラル)
```

【insertRecord】

insertRecord 関数は DB テーブルに新しいレコードを追加（挿入）します。

```
api.dbrequest.insertRecord(appName, target, tableName, data, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
tableName … DBテーブル名
data     … 新しいレコードを表す {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第1引数にはnull、第2引数には以下のプロパティから成るオブジェクトが渡されます。

```
body     … 更新されたレコードの配列。各レコードは {カラム名1: 値1, カラム名2: 値2, ...} の形式のオブジェクト。
```

【outputCrossData】

outputCrossData 関数は DB テーブル・ビューにクロス集計を適用して得られたレコードをファイルに書き出します。

```
api.dbrequest.outputCrossData(appName, target, crossOptions, filters, format, fileName, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
crossOptions … クロス集計オプション。後述の【クロス集計オプション】の節を参照してください。
filters  … 出力フィルタ。カラム名とフィルタ名の対を {"カラム名1": フィルタ名1, "カラム名2": フィルタ名2, …} の形式のオブジェクトで与えます。フィルタについては後述の【フィルタAPI】の節を参照してください。
format   … 出力ファイル形式。現在は "xlsx"（Excelファイル）のみサポートしています。
fileName … 出力ファイル名。省略するとDBテーブル・ビュー名がファイル名として用いられます。
callback … コールバック関数
```

【outputData】

outputData 関数は DB テーブル・ビューのレコードをファイルに書き出します。

```
api.dbrequest.outputData(appName, target, tableName, columns, where, offset, num, order, filters, format, fileName, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
tableName … DBテーブル・ビュー名
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
offset   … 先頭レコード番号(最初のレコードが0)。
num      … 読み込み件数。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
filters  … 出力フィルタ。カラム名とフィルタ名の対を {"カラム名1": フィルタ名1, "カラム名2": フィルタ名2, …} の形式のオブジェクトで与えます。フィルタについては後述の【フィルタAPI】の節を参照してください。
format   … 出力ファイル形式。現在は "xlsx"(Excelファイル)のみサポートしています。
fileName … 出力ファイル名。省略するとDBテーブル・ビュー名がファイル名として用いられます。
callback … コールバック関数
```

【readCachedData】

readCachedData 関数はキャッシュを用いて DB テーブル・ビューからデータ行を読み込みます。以前に読み込んだデータ行がキャッシュに残っていたらそれを実行結果として返します。この場合、サーバへのデータベース処理要求が省略されるので実行が比較的短時間で完了します。

```
api.dbrequest.readCachedData(appName, target, tableName, where, offset, num, order, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
tableName … DBテーブル・ビュー名
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト(err.response.text にエラー内容)が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には得られたレコードの配列が渡されます。各レコードはカラム名と値の対から成る { カラム名 1: 値 1, カラム名 2: 値 2, ...} の形式のオブジェクトで表されます。

【readCrossData】

readCrossData 関数は DB テーブル・ビューにクロス集計を適用して得られたレコードを返します。

```
api.dbrequest.readCrossData(appName, target, crossOptions, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
crossOptions … クロス集計オプション。後述の【クロス集計オプション】の節を参照してくださ
        い。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には得られたレコードの配列が渡されます。各レコードはカラム名と値の対から成る { カラム名 1: 値 1, カラム名 2: 値 2, ...} の形式のオブジェクトで表されます。

【readData】

readData 関数は DB テーブル・ビューからデータ行を読み込みます。

```
api.dbrequest.readData(appName, target, tableName, where, offset, num, order,
 callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
tableName … DBテーブル・ビュー名
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
offset   … 先頭レコード番号（最初のレコードが0）。
num      … 読み込み件数。
order    … 並べ替え。後述の【並べ替え】の節を参照してください。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には得られたレコードの配列が渡されます。各レコードはカラム名と値の対から成る { カラム名 1: 値 1, カラム名 2: 値 2, ...} の形式のオブジェクトで表されます。

【removeListener】

removeListener 関数は前述の addListener 関数で追加したリスナーを削除します。

```
api.dbrequest.removeListener(appName, tableName, listener, callback)
```

引数：

```
appName  … アプリケーション名
tableName … DBテーブル名
listener … リスナー関数
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したら第 1 引数には null が渡されます。第 2 引数は今のところ使用されません。

【setCacheOptions】

setCacheOptions 関数は前述の readCachedData 関数で用いるキャッシュのオプションを設定します。

```
api.dbrequest.setCacheOptions(options)
```

引数：

```
options  … キャッシュオプション。以下のプロパティから成るオブジェクトを与えます。
        size      … キャッシュに保存するreadCachedData関数の実行結果の最大数
```

【updateRecord】

updateRecord 関数は DB テーブルに格納された既存レコードを更新します。

```
api.dbrequest.updateRecord(appName, target, tableName, where, data, callback)
```

引数：

```
appName  … アプリケーション名
target   … ビルド対象("debug"または"release")
tableName … DBテーブル名
where    … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
data     … 更新されるカラムと新しい値の対を表す {カラム名1: 値1, カラム名2: 値2,
           ...} の形式のオブジェクト。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
body     … 更新されたレコードの配列。各レコードは {カラム名1: 値1, カラム名2: 値2,
           ...} の形式のオブジェクト。
```

【upsertRecord】

upsertRecord 関数はレコードの更新を試みて成功したら処理を終えます。もし更新するレコードが無ければ新規レコードの挿入を行ないます。

```
api.dbrequest.upsertRecord(appName, target, tableName, where, data, callback)
```

引数：

```
appName   … アプリケーション名
target    … ビルド対象("debug"または"release")
tableName … DBテーブル名
where     … 絞り込み条件。後述の【絞り込み条件】の節を参照してください。
data      … 更新・挿入されるレコードのカラムと値の対を表す {カラム名1: 値1, カラム名2:
            値2, ...} の形式のオブジェクト。
callback … コールバック関数
```

callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。データ操作が成功した場合、コールバック関数の第 1 引数には null、第 2 引数には以下のプロパティから成るオブジェクトが渡されます。

```
body      … 更新・挿入されたレコードの配列。各レコードは {カラム名1: 値1, カラム名2:
            値2, ...} の形式のオブジェクト。
```

【api.cookie】

api.cookie モジュールはアプリケーションに関連付けられたクッキーを参照するためのモジュールです。クッキーはキーと値の対の並びとして表されます。

【プロパティ】

api.cookie モジュールはクッキーの各々のキーをプロパティ名、キーの値をプロパティの値とするオブジェクトです。キーに含まれる空白文字はプロパティ名からは取り除かれます（たとえば "foo bar" というキーは "foobar" というプロパティになります）。

【api.filter】

api.filter モジュールは Buddy のフィルタ機能を JavaScript コードから利用するための API 関数を提供するモジュールです。

【関数】

【apply】

apply 関数は与えられたデータにフィルタを適用した結果を返します。

```
api.filter.apply(filterName, data)
```

引数 :

```
filterName … フィルタ名
data       … フィルタ適用対象のデータ(文字列など)
```

filterName 引き数には適用するフィルタの名前を文字列で与えます。利用できるフィルタ名については【フィルタ API】の節を参照してください。

【exists】

exists 関数は与えられた名前のフィルタが存在するかどうかを返します。

```
api.filter.exists(filterName)
```

引数 :

```
filterName … フィルタ名
```

filterName 引数で与えた名前のフィルタが存在するならば真を、それ以外の場合は偽を返します。

【set】

set 関数は新しいフィルタを登録します。

```
api.filter.set(filterName, func)
```

引数 :

```
filterName … フィルタ名
func       … 関数オブジェクト
```

filterName 引数には新しいフィルタ名を、func 引数にはフィルタを実装する関数オブジェクトを与えます。

【api.outputDialog】

api.outputDialog モジュールはレポート出力を実行してレポートをダウンロードするためのダイアログを表示するモジュールです。

【関数】

【open】

open 関数はレポート出力ダイアログを開きます。

```
api.outputDialog.open(outputName, param)
```

引数：

```
outputName … レポート名
param      … レポートの制御パラメタ
```

outputName 引数にはレポート設計で定義したレポート名を与えます。param 引数にはレポート制御パラメタを与えます。指定したパラメタは api.request.outputReport 関数の param 引数に渡されます。パラメタの詳細については【outputReport】の節を参照してください。

【api.dialog】

api.dialog モジュールは各種のダイアログウィンドウを表示するための API 関数を提供するモジュールです。

【関数】

【alert】

alert 関数はメッセージを警告用ダイアログで表示します。

```
api.dialog.alert(message, callback)
```

引数：

```
message  … メッセージ文字列
callback … コールバック関数
```

message 引数にはダイアログ内に表示するメッセージを文字列で与えます。このダイアログは閉じられるまで JavaScript コードの実行をブロックします。コールバック関数はダイアログが閉じられた後に呼び出されます。

【closeDialog】

closeDialog 関数は後述の showModal 関数で開いたダイアログを閉じます。

```
api.dialog.closeDialog(code, value)
```

引数：

```
code     … 終了コード
value    … ダイアログの戻り値
```

code 引数にはダイアログの閉じ方を示す任意の値を与えます。value 引数にはダイアログからスクリーンプログラムに渡す戻り値を与えます。code 引数と value 引数の値はそれぞれ showModal 関数で開いたコールバック関数の第 1 引数、第 2 引数に渡されます。

【confirm】

confirm 関数はメッセージを確認用ダイアログで表示します。ダイアログウィンドウには OK ボタンとキャンセルボタンが表示されます。

```
api.dialog.confirm(message, callback)
```

引数：

```
message  … メッセージ
callback … コールバック関数
```

message 引数にはダイアログ内に表示するメッセージを文字列で与えます。このダイアログは閉じられるまで JavaScript コードの実行をブロックします。コールバック関数は

ダイアログが閉じられた後に呼び出されます。コールバック関数の引数には、OK ボタンが押されたら true、キャンセルボタンが押されたら false が渡されます。

【message】

message はメッセージをアプリケーション内のダイアログで表示します。

```
api.dialog.message(message, style, callback)
```

引数：

```
message  … メッセージ文字列
style    … スタイルオブジェクト
callback … コールバック関数
```

message 引数にはダイアログ内に表示するメッセージを文字列で与えます。style 引数にはスタイルオブジェクトを与えます。このダイアログは JavaScript コードの実行をブロックしません。ダイアログが閉じられたらコールバック関数が呼び出されます。

【openFile】

openFile 関数はアプリケーションのファイルシステム内のファイルをブラウザの別ウィンドウに表示します。

```
api.dialog.openFile(path, target)
```

引数：

```
path     … ファイル名
target   … ウィンドウ名
```

path 引数にはファイル名を /files ディレクトリからの相対パスで指定します。target 引数にはブラウザのウィンドウ名を与えます。指定した名前のウィンドウがあればそのウィンドウが用いられ、なければその名前の新しいウィンドウが開かれます。target 引数に "_blank" を指定すると常に新しいウィンドウが開かれます。target 引数を省略すると "_blank" を指定したと見なされます。

【openUrl】

openUrl 関数は URL で指定されるリソース（ウェブページなど）をブラウザの別ウィンドウに表示します。

```
api.dialog.openUrl(url, target)
```

引数：

```
url      … リソースのURL
target   … ウィンドウ名
```

url 引数には表示するリソースの URL を文字列で与えます。target 引数にはブラウザのウィンドウ名を与えます。指定した名前のウィンドウがあればそのウィンドウが用いら

れ、なければその名前の新しいウィンドウが開かれます。target引数に"_blank"を指定すると常に新しいウィンドウが開かれます。target 引数を省略すると "_blank" を指定したと見なされます。

【show】

show 関数はコンテナ型のスクリーンをアプリケーション内のダイアログに埋め込んで表示します。

```
api.dialog.show(screen, style, callback)
```

引数：

```
screen   … スクリーン名
style    … スタイルオブジェクト
callback … コールバック関数
```

screen 引数にはダイアログに埋め込むスクリーンの名前を与えます。指定するスクリーンはコンテナ型でなければなりません。style 引数にはスタイルオブジェクトを与えます。このダイアログはJavaScriptコードの実行をブロックしません。show関数で開いたダイアログは、埋め込みスクリーンのスクリーンプログラムの中で前述の closeDialog 関数を呼び出すか、ダイアログ右上の X ボタンまたはダイアログの外側をクリックすれば閉じることができます。ダイアログが閉じられたらコールバック関数が呼び出されます。コールバック関数の引数にはダイアログの閉じ方に応じて以下の値が渡されます。

```
a.  closeDialog関数によりダイアログを閉じたとき
      closeDialog関数のcode引数、value引数に与えた値がそれぞれコールバック関数の第1引数、第2引数に渡されます。
b.  ダイアログ右上のXボタンまたはダイアログの外側をクリックして閉じたとき
      コールバック関数の第1引数にはnullが渡されます。第2引数の値は未定義です。
```

【showDateSelector】

showDateSelector は日付選択ダイアログを表示します。

```
api.dialog.showDateSelector(options, callback)
```

引数：

```
options  … オプション
callback … コールバック関数
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
date     … 日付選択の初期値(Date型オブジェクトまたはDate型に変換できる日付文字列)
style    … スタイルオブジェクト
```

このダイアログは JavaScript コードの実行をブロックしません。ダイアログが閉じられたらコールバック関数が呼び出されます。コールバック関数の引数には選択された日付が Date 型で渡されます。ダイアログの外側をクリックしてダイアログを閉じた場合、

コールバック関数に渡される日付は未定義になります。

【showFileSelector】

showFileSelector はファイル選択ダイアログを表示します。

```
api.dialog.showFileSelector(options, callback)
```

引数：

```
options  … オプション
callback … コールバック関数
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
defaultValue … ファイル名入力欄のデフォルト値
deletable … 真ならファイルの削除を許可する
downloadable … 真ならファイルのダウンロードを許可する
filterList … ファイル名フィルタの配列。各要素は {name: 名前, format: ファイル名の正規
        表現} の形式のオブジェクト
movable  … 真ならファイルの移動を許可する
multiple … 真ならファイルの複数選択を許可する(デフォルトは偽)
rootDir  … ルートディレクトリ(デフォルトはfiles)
rootURL  … ルートURL(デフォルトは/develop/アプリケーション名/files)
uploadable … 真ならファイルのアップロードを許可する
```

このダイアログは JavaScript コードの実行をブロックしません。ダイアログが閉じられたらコールバック関数が呼び出されます。コールバック関数の引数には選択されたファイル名(multiple オプションが真ならファイル名の配列)が渡されます。ダイアログの外側をクリックしてダイアログを閉じた場合、コールバック関数に渡されるファイル名は未定義になります。filterList オプションにはファイル名フィルタの配列を与えます。ファイル名フィルタの name プロパティはフィルタ選択メニューの表示に用いられ、format プロパティは指定した正規表現にマッチするファイル名のみを表示するために用いられます。filterList オプションを省略すると [{name: " すべて ", format: ".*"}] が与えられたと見なされます。rootDir オプションは指定されたディレクトリ配下からのみファイル名を選択できるようにするために用います。rootURL オプションにはファイルをダウンロードするときのルート URL を与えます。

【showModal】

showModal 関数はコンテナ型のスクリーンをアプリケーション内のダイアログに埋め込んで表示します。

```
api.dialog.showModal(screen, style, callback)
```

引数：

```
screen   … スクリーン名
style    … スタイルオブジェクト
```

```
callback … コールバック関数
```

screen 引数にはダイアログに埋め込むスクリーンの名前を与えます。指定するスクリーンはコンテナ型でなければなりません。style 引数にはスタイルオブジェクトを与えます。このダイアログは JavaScript コードの実行をブロックしません。showModal 関数で開いたダイアログは、埋め込みスクリーンのスクリーンプログラムの中で前述の closeDialog 関数を呼び出すことで閉じることができます。ダイアログが閉じられたらコールバック関数が呼び出され、closeDialog 関数の code 引数、value 引数に与えた値がそれぞれコールバック関数の第 1 引数、第 2 引数に渡されます。

【showQuestion】

showQuestion 関数は質問ダイアログを表示します。

```
api.dialog.showQuestion(options, callback)
```

引数：

```
options  … オプション
callback … コールバック関数
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
choices  … 選択肢の配列。各要素は {label: ラベル, value: 値} の形式のオブジェクト。
message  … メッセージ文字列
style    … スタイルオブジェクト
```

このダイアログは JavaScript コードの実行をブロックしません。ダイアログが閉じられたらコールバック関数が呼び出されます。コールバック関数の引数には選択された options.choices の要素の value プロパティの値が渡されます。ダイアログの外側をクリックしてダイアログを閉じた場合、コールバック関数に渡される値は未定義になります。

【api.store】

api.store モジュールはスクリーン間でデータを共有する仕組みを提供するモジュールです。api.store モジュールの実体は空のオブジェクトです。このオブジェクトはスクリーン間でデータを共有するためのキーバリューストアとして自由に利用できます。

たとえば、api.store モジュールとスクリーンオブジェクトの serialize メソッドおよび deserialize メソッドを組み合わせて、別のスクリーンに移って戻ってきたときにスクリーンの状態を復元する機能を実現することができます。この機能のプログラム例については【serialize】の節を参照してください。

api.store モジュールに保存されたデータには永続性はなく、ブラウザを閉じたりアプリケーションをリロードしたりすると失われます。永続性が必要な場合は次節の api.KeyValueStore モジュールを利用してください。

【api.KeyValueStore】

api.KeyValueStore モジュールは永続性のあるキーバリューストアを提供するモジュールです。デバッグ用アプリとリリース用アプリはそれぞれ別のキーバリューストアを有します。

【関数】

【get】

get 関数はキーバリューストアからキーの値を取得します。

```
api.KeyValueStore.get(key, callback)
```

引数：

```
key      … キー
callback … コールバック関数
```

key 引数にはキーを与えます。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数にはキーの値が渡されます。実行が失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【set】

set 関数はキーバリューストアに新しいキーと値の対を追加、または既存のキーの値を変更します。

```
api.KeyValueStore.set(key, value, callback)
```

引数：

```
key      … キー
value    … 値
callback … コールバック関数
```

key 引数にはキー、value 引数にはそのキーの値を与えます。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。実行が成功したらコールバック関数の第 1 引数には null、第 2 引数にはキーの値が渡されます。実行が失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【api.serverFunction】

api.serverFunction モジュールはサーバー機能設計で定義した関数（これをサーバー関数と呼びます）を実行するための仕組みを提供するモジュールです。

【関数】

【execute】

execute 関数はサーバー関数を同期実行します。サーバー側の関数実行プロセスは関数の実行完了までブロックします。

```
execute(funcName, options, callback)
```

引数：

```
funcName … 関数名
options  … オプション
callback … コールバック関数
```

funcName 引数には実行するサーバー関数の名前を与えます。options 引数にはサーバー関数のオプションを与えます。オプションはそのままサーバー関数に渡されます。オプションのデータ構造はサーバー関数の都合に応じて自由に決めることができます。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。サーバー関数の実行が成功したらコールバック関数の第 1 引数には null、第 2 引数にはサーバー関数からの戻り値が渡されます。実行が失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【executeAsync】

executeAsync 関数はサーバー関数を非同期実行します。サーバー側の関数実行プロセスは関数の実行完了を待たずに処理を継続します。

```
executeAsync(funcName, options, callback)
```

引数：

```
funcName … 関数名
options  … オプション
callback … コールバック関数
```

funcName 引数には実行するサーバー関数の名前を与えます。options 引数にはサーバー関数のオプションを与えます。オプションはそのままサーバー関数に渡されます。オプションのデータ構造はサーバー関数の都合に応じて自由に決めることができます。callback 引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。サーバー関数の実行が成功したらコー

ルバック関数の第 1 引数には null、第 2 引数にはサーバー関数からの戻り値が渡されます。実行が失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【api.TextSearch】

api.TextSearch クラスはデータベースの全文検索に用いる絞り込み条件をより直感的で平易な検索式から構築します。たとえば、address カラムに「東京」または「大阪」を含むレコードを得るための絞り込み条件は

```
{ "OR": [
    { "address": { op: "LIKE", value: "東京" } },
    { "address": { op: "LIKE", value: "大阪" } } ] }
```

のように記述する必要があります。これに対して api.TextSearch モジュールでは

```
東京 OR 大阪
```

という検索式を与えれば前掲の絞り込み条件に相当するオブジェクトが作成できます。例：

```
var textsearch = new api.TextSearch();
textsearch.parse("東京 AND 大阪");
var where = textsearch.createWhere(["address"]);
```

このコード例の処理内容は次のとおりです。まず 1 行目で TextSearch オブジェクト（TextSearch クラスのインスタンス）を作成します。次に 2 行目で検索式の構成が正しいかどうかをチェックします。最後に 3 行目でカラム名を与えて絞り込み条件のオブジェクトを作成します。

【メソッド】

【parse】

parse メソッドは検索式の構成を検証します。

```
TextSearch.parse(expr)
```

引数：

```
expr      … 検索式（後述）
```

与えられた検索式が正しいときは true、そうでなければ false を返します。parse メソッドは内部では構文解析器を用いて検索式を構文木に変換します。得られた構文木は TextSearch オブジェクト内で保存されて createWhere メソッドの呼び出し時に絞り込み条件オブジェクトを作成するのに用いられます。

【createWhere】

createWhere メソッドは検証済みの検索式から絞り込み条件のオブジェクトを作成します。このメソッドは parse メソッドで検索式を検証してから呼び出さなければなりません。

```
TextSearch.createWhere(columns)
```

引数：

```
columns　… DBテーブル・ビューのカラム情報の配列
```

columns 引数には以下のいずれかの形式のカラム情報を要素とする配列を与えます。

```
a.　カラム名文字列
b.　ModelTable.conf.TableColumns配列の要素（【ModelTable】の節を参照）
c.　ModelView.conf.ViewColumns配列の要素（【ModelView】の節を参照）
```

【api.lib.async】

api.lib.async モジュールは非同期処理の実行順序を制御するための機能を提供するモジュールです。本節では async モジュールの中で特に多用される関数について説明します。async モジュールの網羅的なドキュメントは下記 URL で提供されています。

```
http://caolan.github.io/async/docs.html
```

【parallel】

parallel 関数は、複数の関数を並行して実行します。

```
api.lib.async.parallel(tasks, [callback])
```

引数：

```
tasks    … タスク関数の配列
callback … コールバック関数
```

tasks には関数の配列を与えます（これらの関数をタスク関数と呼びます）。これらのタスク関数は、他のタスク関数が実行中かどうかに関わらず並行して順不同で実行されます。各タスク関数にはコールバック関数が引数として渡されます。タスク関数の実行終了時にはこのコールバック関数を呼び出さなければなりません。実行が成功ならコールバック関数の第 1 引数には null、第 2 引数には実行結果を与え、実行が失敗なら第 1 引数にエラーオブジェクトを与えます。いずれかのタスク関数の実行が失敗したら、parallel 関数の callback 引数に与えたコールバック関数がただちに呼び出され、その引数にはタスク関数から返されたエラーオブジェクトが渡されます。すべてのタスク関数の実行が成功したら、parallel 関数のコールバック関数の第 1 引数には null、第 2 引数にはタスク関数から返された実行結果の配列が渡されます。

```
api.lib.async.parallel([
    function(callback){
        // 何らかの処理を実行する
        callback(null, 'one');
    },
    function(callback){
        // 並行して何らかの処理を実行する
        callback(null, 'two');
    }
],
// コールバック関数(省略可)
function(err, results){
    // results引数の値は ['one', 'two']
});
```

【series】

series 関数は、1 つ前の関数の実行が終わるのを待って次の関数の実行に移るという形で複数の関数を逐次実行します。

```
api.lib.async.series(tasks, [callback])
```

引数：

```
tasks    … タスク関数の配列
callback … コールバック関数
```

tasks には関数の配列を与えます（これらの関数をタスク関数と呼びます）。これらのタスク関数は、1 つ前のタスク関数の実行が終わってから次のタスク関数の実行に移るという形で実行されます。各タスク関数にはコールバック関数が引数として渡されます。タスク関数の実行終了時にはこのコールバック関数を呼び出さなければなりません。実行が成功ならコールバック関数の第 1 引数には null、第 2 引数には実行結果を与え、実行が失敗なら第 1 引数にエラーオブジェクトを与えます。いずれかのタスク関数にエラーが渡されたら、それ以降のタスク関数は実行されず、callback 引数に与えたコールバック関数がただちに実行されます。このコールバック関数の第 1 引数にはタスク関数からのエラーオブジェクトが渡されます。すべてのタスク関数が正常終了したら、callback 引数で与えられたコールバック関数の第 1 引数には null、第 2 引数にはタスク関数からの実行結果の配列が渡されます。

```
api.lib.async.series([
    function(callback){
        // 何らかの処理を実行する
        callback(null, 'one');
    },
    function(callback){
        // さらに何らかの処理を実行する
        callback(null, 'two');
    }
],
// コールバック関数(省略可)
function(err, results){
    // results引数の値は ['one', 'two']
});
```

【waterfall】

waterfall 関数は、1 つ前の関数の実行結果を次の関数の入力値として渡すという形で複数の関数を逐次実行します。

```
api.lib.async.waterfall(tasks, [callback])
```

引数：

```
tasks    … タスク関数の配列
callback … コールバック関数
```

tasks には関数の配列を与えます（これらの関数をタスク関数と呼びます）。これらのタスク関数は、1 つ前のタスク関数の実行結果が次のタスク関数の入力値として渡される形で逐次実行されます。各タスク関数にはコールバック関数が引数として渡されます。タスク関数の実行終了時にはこのコールバック関数を呼び出さなければなりません。実行が成功ならコールバック関数の第 1 引数には null、第 2 引数以降には実行結果を与え、実行が失敗なら第 1 引数にエラーオブジェクトを与えます。いずれかのタスク関数にエラーが渡されたら、それ以降のタスク関数は実行されず、callback 引数に与えたコールバック関数がただちに実行されます。このコールバック関数の第 1 引数にはタスク関数からのエラーオブジェクトが渡されます。すべてのタスク関数が正常終了したら、callback 引数で与えられたコールバック関数の第 1 引数には null、第 2 引数には最後のタスク関数の実行結果が渡されます。

```
api.lib.async.waterfall([
    function(callback) {
        callback(null, 'one', 'two');
    },
    function(arg1, arg2, callback) {
        // arg1引数の値は 'one'、arg2引数の値は 'two'
        callback(null, 'three');
    },
    function(arg1, callback) {
        // arg1引数の値は 'three'
        callback(null, 'done');
    }
], function (err, result) {
    // result引数の値は 'done'
});
```

【api.lib.objectAssign】

api.lib.objectAssign 関数は対象オブジェクトに別のオブジェクトのプロパティを代入して新しいオブジェクトを返します。

```
api.lib.objectAssign(target, source, [source, ...])
```

target 引数には対象オブジェクト、source 引数には別のオブジェクトを与えます。source 引数を複数与えた場合、同名のプロパティは後から与えた source 引数のオブジェクトのプロパティ値で上書きされます。

【api.lib.superagent】

api.lib.superagent モジュールは HTTP クライアントを簡便に作成するための API を提供するモジュールです。本節では superagent モジュールの中で特に多用される API 関数について説明します。superagent モジュールの網羅的なドキュメントは下記 URL で提供されています。

```
http://visionmedia.github.io/superagent/
```

たとえば GET メソッドでリソース URL /search の内容を得るには以下のようにします。

```
api.lib.superagent
  .get('/search')
  .end(function(err, res){
      if (err || !res.ok) {
          alert('エラー');
      } else {
          alert('結果:' + JSON.stringify(res.body));
      }
  });
```

【関数】

【end】

end 関数はリクエストを送信します。引数にはレスポンスを処理するコールバック関数を与えます。次の例では GET リクエストを end 関数で送信して、引数に与えたコールバック関数でレスポンスを受け取ります。

```
api.lib.superagent
  .get('/search')
  .end(function(err, res){
      // ここにレスポンスの処理を記述する
  });
```

【get】

get 関数は GET リクエストの作成を開始します。引数にはリソース URL を文字列で与えます。次の例では get 関数でリソース URL /search への GET リクエストの作成を開始しています。

```
api.lib.superagent
  .get('/search')
  .end(callback);
```

【head】

head 関数は HEAD リクエストの作成を開始します。引数にはリソース URL を文字列で与えます。次の例では head 関数でリソース URL /favicon.ico への HEAD リクエストの作成を開始しています。

```
api.lib.superagent
  .head('/favicon.ico')
  .end(callback);
```

【post】

post 関数は POST リクエストの作成を開始します。引数にはリソース URL を文字列で与えます。次の例では post 関数でリソース URL /user/add への POST リクエストの作成を開始しています。

```
api.lib.superagent
  .post('/user/add')
  .send({ name: 'john', age: 20 })
  .end(callback);
```

【query】

query 関数はリクエストの URL にクエリを追加します。引数にはクエリをオブジェクトまたは文字列で与えます。次の例では GET リクエストの URL にクエリを追加して /search?q=test&order=desc にします。

```
api.lib.superagent
  .get('/search')
  .query({'q': 'test', 'order', 'desc'})
  .end(callback);
```

query 関数は複数回に分けて呼び出すこともできます。

```
api.lib.superagent
  .get('/search')
  .query({'q': 'test'})
  .query({'order', 'desc'})
  .end(callback);
```

クエリは文字列でも指定できます。

```
api.lib.superagent
  .get('/search')
  .query('q=test&order=desc')
  .end(callback);
```

文字列の場合も複数に分けて与えることができます。

```
api.lib.superagent
  .get('/search')
  .query('q=test')
  .query('order=desc')
  .end(callback);
```

【send】

send 関数は作成中のリクエストにデータを追加します。次の例では send 関数で POST リクエストにデータを追加しています。

```
api.lib.superagent
  .post('/user/add')
  .send({ name: 'john', age: 20 })
  .end(callback);
```

send 関数は複数回に分けて呼び出すこともできます。

```
api.lib.superagent
  .post('/user/add')
  .send({ name: 'john' })
  .send({ age: 20 })
  .end(callback);
```

データの型を示す Content-Type: ヘッダフィールドのデフォルト値は send 関数の引数によって決まります。

```
a.  引数がオブジェクトの場合:application/json
b.  引数が文字列の場合:application/x-www-form-urlencoded
```

Content-Type: ヘッダフィールドの値を設定するには set 関数を用います。次の例は上述の最初の例と同等です。

```
api.lib.superagent
  .post('/user/add')
  .set('Content-Type', 'application/json')
  .send({ name: 'john', age: 20 })
  .end(callback);
```

【set】

set 関数は作成中のリクエストにヘッダフィールドを追加します。次の例では GET リクエストに Accept: application/json というヘッダフィールドを追加します。

```
api.lib.superagent
  .get('/search')
  .set('Accept', 'application/json')
  .end(callback);
```

【api.lib.EventEmitter】

api.lib.EventEmitter クラスは Node.js でイベント駆動型のアプリケーションを開発するためのイベント API を提供するクラスです。本節では EventEmitter の主な機能について説明します。EventEmitter の網羅的なドキュメントは下記 URL で提供されています。

```
https://nodejs.org/api/events.html
```

EventEmitter は以下の例のようにインスタンスを作成して利用します。

```
var ev = new api.lib.EventEmitter();
```

イベントは EventEmitter の emit メソッドを実行することで発生します。EventEmitter のアプリケーションプログラムは、イベント発生の通知を受け取るために EventEmitter の on メソッドでリスナー関数を登録します。次の例では、まず on メソッドで done イベントのリスナー関数を登録しています。次に、emit メソッドで done イベントを生成しています。emit 呼び出し時の第 2 引数以降はリスナー関数に引数として渡されます。

```
var ev = new api.lib.EventEmitter();
ev.on('done', function (arg1, arg2) {
    console.log(arg1, arg2);
});
ev.emit('done', 123, 'abc');
// コンソールに 123 abc と表示される
```

イベントは時間のかかる処理の完了を待って別の処理を続けたいときに多用されます。次の例では、done イベントの発生に 5000 ミリ秒かかります。done イベントのリスナー関数はイベント発生まで待機します。

```
var ev = new api.lib.EventEmitter();
setTimeout(function () {
    ev.emit('done', 123, 'abc');
}, 5000);
ev.on('done', function (arg1, arg2) {
    console.log(arg1, arg2);
});
```

on メソッドで登録されたリスナー関数はイベントが発生するたびに繰り返し実行されます。一度だけ実行されるリスナー関数を登録するには once メソッドを用います。例:

```
var ev = new api.lib.EventEmitter();
ev.once('data, function (result) {
    console.log('start');
});
ev.on('data, function (result) {
    console.log(result);
});
ev.emit('data', 123);
// コンソールに start と表示される
```

```
// コンソールに 123 と表示される
ev.emit('data', 'abc');
// コンソールに abc と表示される
```

【メソッド】

本節では EventEmitter クラスの主なメソッドについて説明します。

【addListener】

addListener メソッドは指定されたイベントに対するリスナー関数を登録します。このメソッドは EventEmitter インスタンス自身を返します。

```
addListener(event, listener)
```

event 引数にはイベント名、listener 引数には登録するリスナー関数を与えます。

【emit】

emit メソッドは指定されたイベントに対して登録されているリスナー関数を実行します。リスナー関数があれば true、そうでなければ false を返します。

```
emit(event[, arg1[, arg2[, ...]]])
```

event 引数にはイベント名を与えます。追加の引数 arg1、arg2、…（省略可）はリスナー関数に渡されます。

【listeners】

listeners メソッドは指定されたイベントに対して登録されているのリスナー関数を配列で返します。

```
listeners(event)
```

event 引数にはイベント名を与えます。

【on】

on メソッドは前述の addListener メソッドの別名です。詳しくは【addListener】の節を参照してください。

```
on(event, listener)
```

【once】

once メソッドは一度だけ実行されるリスナー関数を登録します。このメソッドは EventEmitter インスタンス自身を返します。

```
once(event, listener)
```

event 引数にはイベント名、listener 引数には登録するリスナー関数を与えます。

【removeAllListeners】

removeAllListeners メソッドはすべてのリスナー関数、または指定されたイベントのすべてのリスナー関数の登録を解除します。このメソッドは EventEmitter インスタンス自身を返します。

```
removeAllListeners([event])
```

event 引数にはイベント名を与えます（省略可）。

【removeListener】

removeListener メソッドは指定されたイベントに対するリスナー関数の登録を解除します。このメソッドは EventEmitter インスタンス自身を返します。

```
removeListener(event, listener)
```

event 引数にはイベント名、listener 引数には登録解除するリスナー関数を与えます。

【api.lib.Decimal】

api.lib.Decimal モジュールは誤差のない 10 進数演算を行なうための API を提供します。JavaScript の数値（Number オブジェクト）には誤差が含まれることがあるのに対して、Decimal オブジェクトは誤差のない四則演算を実現できます。次の例では 4.01 と 2.01 の積を 10 進数演算で計算します。

```
api.lib.Decimal('4.01').mul('2.01').toNumber()
```

上記のコードは次のように記述することもできます。

```
api.lib.Decimal.mul('4.01', '2.01').toNumber()
```

【関数】

【Decimal】

新しい Decimal オブジェクトを作成します。

```
api.lib.Decimal(n)
```

引数 n には文字列、整数、または別の Decimal オブジェクトを与えます。

【メソッド】

この節では Decimal オブジェクトのメソッドについて説明します。

【toString】

Decimal オブジェクトを文字列にして返します。

```
toString()
```

【toNumber】

Decimal オブジェクトを数値（Number オブジェクト）にして返します。

```
toNumber()
```

【add】

Decimal オブジェクトと引数で与えられた値の和を新しい Decimal オブジェクトにして返します。

```
add(n)
```

引数 n には文字列、整数、または別の Decimal オブジェクトを与えます。

【sub】

Decimal オブジェクトと引数で与えられた値の差を新しい Decimal オブジェクトにして返します。

```
sub(n)
```

引数 n には文字列、整数、または別の Decimal オブジェクトを与えます。

【mul】

Decimal オブジェクトと引数で与えられた値の積を新しい Decimal オブジェクトにして返します。

```
mul(n)
```

引数 n には文字列、整数、または別の Decimal オブジェクトを与えます。

【div】

Decimal オブジェクトを引数で与えられた値で除した値を新しい Decimal オブジェクトにして返します。

```
div(n)
```

引数 n には文字列、整数、または別の Decimal オブジェクトを与えます。

【api.lib.xml2js】

api.lib.xml2js モジュールは XML を読み取る API を提供します。

```
var xml = '<root>Hello</root>';
api.lib.xml2js.parseString(xml, function (err, data) {
    console.log(data);
});
```

【関数】

【parseString】

parseString 関数は与えられた XML データを解析します。

```
parseString(xml, callback)
```

引数：

```
xml      … XMLデータ
callback … コールバック関数
```

xml引数には解析するする XML データを文字列で与えます。callback引数には実行結果を受け取るコールバック関数を指定します。コールバック関数を callback(err, res) とすると、実行が失敗したら第 1 引数に Error オブジェクト（err.response.text にエラー内容）が返されます。解析が成功したらコールバック関数の第1引数にはnull、第2引数には解析結果が渡されます。解析が失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【データベース API】

DB テーブル･ビューのデータ操作に関する API 関数では、データ操作オプションを共通の方法で指定します。データ操作オプションは原則として以下のプロパティから成るオブジェクトで与えられます。

```
where    … 絞り込み条件
order    … 並べ替え
offset   … 開始レコード番号(最初のレコードが0)
num      … レコード数
crossOptions … クロス集計オプション
```

このうち本節では絞り込み条件、並べ替え、クロス集計オプションについて述べ、絞り込み条件で用いるパタン文字列について説明します。また、絞り込み条件および結合に用いるサブクエリ（副問い合わせ）について述べます。

【絞り込み条件】

絞り込み条件（where）は DB テーブル･ビューに格納されたレコードのうち、読み出しや更新などのデータ操作の対象となるレコードを選択するための条件を記述するオプションです。絞り込み条件を指定すると、その条件を満たすレコードのみが処理対象となります。絞り込み条件を指定しなければ DB テーブル・ビューの全レコードが処理対象となります。絞り込み条件として指定できるオブジェクトは以下のとおりです。

```
1. カラム名と値の対から成るオブジェクト
2. サブクエリ(副問い合わせ)を含むオブジェクト
3. 絞り込み条件の論理和･論理積を表すオブジェクト
```

【カラム名と値の対】

絞り込み条件としてカラム名と値の対から成る次の形式のオブジェクトを与えた場合、カラムの値が所与の関係を満たすかどうかをすべての対について調べます。

```
{"カラム名1": カラム1の値, "カラム名2": カラム2の値, ...}
```

カラムの値として指定できるのは以下のものです。

```
a. 数値や文字列などのリテラル(即値)
b. {op: 演算子, value: 値} の形式のオブジェクト
c. {op: 演算子, values: [値1, 値2, ...]} の形式のオブジェクト
d. {op: 演算子, min: 下限値, max: 上限値} の形式のオブジェクト
e. {op: 演算子} の形式のオブジェクト
```

【リテラル】

値として数値や文字列などのリテラル（即値）を指定した場合、その値とカラムの値が等しいかどうかを調べます。この指定方法は、後述の比較演算子を用いた形式で {op: "=", value: 値 } と記述するのと同等です。

【比較演算子】

値として次の形式のオブジェクトを与えた場合、カラムの値と value の値を演算子 op で比較します。

```
{op: 演算子, value: 値}
```

op には以下の演算子が指定できます。

```
=       … 等しい
!=      … 等しくない
<>      … 等しくない
<       … より小さい
<=      … より小さい､または等しい(以下)
>       … より大きい
```

```
>=        … より大きい、または等しい（以上）
LIKE      … valueで与えられたパタン文字列に一致する
NOT LIKE … valueで与えられたパタン文字列に一致しない
```

パタン文字列の最初および最後に現れるパーセント記号（%）は 0 個以上の文字の並びと一致します。たとえば {op: "LIKE", value: " 日本 %"} と指定すると " 日本 "、" 日本人 "、" 日本列島 " などに一致します。パタン文字列については後述の【パタン文字列】の節を参照してください。

【IN】

値として次の形式のオブジェクトを与えた場合、カラムの値が values の要素として含まれるかどうかを調べます。

```
{op: 演算子, values: [値1, 値2, ...]}
```

op には以下の演算子が指定できます。

```
IN        … カラムの値がvaluesの要素として含まれる
NOT IN    … カラムの値がvaluesの要素として含まれない
```

values には 1 つ以上の要素からなる配列を与えます。空の配列を与えるとエラーになります。

【BETWEEN】

値として次の形式のオブジェクトを与えた場合、カラムの値が最小値と最大値で表された範囲に含まれるかどうかを調べます。

```
{op: 演算子, min: 最小値, max: 最大値}
```

op には以下の演算子が指定できます。

```
BETWEEN  … カラムの値が範囲内にある（最小値以上かつ最大値以下である）
NOT BETWEEN … カラムの値が範囲外にある（最小値以上でない、または最大値以下でない）
```

【IS NULL】

値として次の形式のオブジェクトを与えた場合、op に指定した条件が成り立つかどうかを調べます。

```
{op: 演算子}
```

op には以下の演算子が指定できます。

```
IS NULL  … カラムの値がNULLである
IS NOT NULL … カラムの値がNULLではない
```

【サブクエリによる絞り込み】

サブクエリを用いた絞り込み条件には以下のものがあります。

a. {カラム名: {op: 演算子, ref: サブクエリ}, ...}の形式のオブジェクト … カラムの値とサブクエリの結果をop:プロパティで指定した演算子で比較します。この場合のサブクエリは結果が1件だけでなければなりません。
b. {カラム名: {op: 演算子, q: ALLまたはANYまたはSOME, ref: サブクエリ}, ...}の形式のオブジェクト … カラムの値とサブクエリの結果をop:プロパティで指定した演算子で比較して、すべて(q:プロパティの値がALLの場合)または1件以上(ANYまたはSOMEの場合)について真かどうかを調べます。
c. {EXISTS: {ref: サブクエリ}}の形式のオブジェクト … サブクエリの結果が1件以上あるか否かを調べます。

ここで ref: プロパティにはサブクエリオブジェクトを与えます。サブクエリオブジェクトについては【サブクエリ】の節を参照してください。

【絞り込み条件の論理和・論理積】

次の形式で絞り込み条件オブジェクトの配列を与えた場合、それらの絞り込み条件の論理和（OR）や論理積（AND）が成り立つかどうかを調べます。

```
{論理演算子: [絞り込み条件1, 絞り込み条件2, ...]}
```

論理演算子には以下のものが指定できます。

```
OR      … 論理積(「または」の意)
AND     … 論理積(「かつ」の意)
```

【用例】

絞り込み条件の用例を以下に示します。

1. 社員番号が 100 に等しいレコードを選択します。

```
{"社員番号": 100}
```

2. 年齢が 20 歳以上のレコードを選択します。

```
{"年齢": {op: ">=", value: 20}}
```

3. 年齢が 20 歳以上で、なおかつ性別が女性のレコードを選択します。

```
{"年齢": {op: ">=", value: 20}, "性別": "女性"}
```

4. 性別が男性または未指定のレコードを選択します。

```
{"性別": {op: "IN", values: ["男性", "未指定"]}}
```

5. 注文日が 2010 年のレコードを選択します。

```
{"注文日": {op: "BETWEEN", min: 2010/01/01", max: "2010/12/31"}}
```

6. 年齢が 20 歳以上、または性別が女性のレコードを選択します。

```
{"OR": [{"年齢": {op: ">=", value: 20}}, {"性別": "女性"}]}
```

また、サブクエリの例として次の SQL 文を考えます。この問い合わせは天気（weather）テーブルを検索して最低気温（temp_lo）カラムの値が最大の都市名（city）カラムの値を返します。

```
SELECT city FROM weather
WHERE temp_lo = (SELECT max(temp_lo) FROM weather);
```

この問い合わせを Buddy で実現するには次のようなスクリーンプログラムを記述します。

```
var tableName = "weather";
var subquery = this.tables[tableName].select(["max(temp_lo)"], {});
var options = {
    where: {"temp_lo": {op: "=", ref: subquery}},
};
this.tables[tableName].readData(options, (function(error, data) {
    ...
});
```

このとき options.where に与えられる絞り込み条件は以下の通りです。

```
{"temp_lo": {op: "=", ref: {select: ['max(temp_lo)'], from: {model: 'weather'}}}}
```

【並べ替え】

並べ替え（order）には以下のプロパティから成るオブジェクトの配列を与えます。

```
OrderColumn … カラム名またはカラム値を用いるSQL式
OrderDirection … ソート順
```

OrderColumn プロパティにはカラム名またはカラム値を用いる SQL 式を文字列で与えます。OrderDirection プロパティにはソート順を「ASC または DESC NULLS FIRST または LAST」の形式の文字列で与えます。ASC は昇順、DESC は降順を表します。NULLS 以降は省略できます。NULLS FIRST は NULL 値をソート結果の先頭に、NULLS LAST は末尾に置くことを指定します。NULLS 以降を省略したときのデフォルトの NULL 値の順序は、ASC の場合は NULLS LAST、DESC の場合は NULLS FIRST です。

【クロス集計オプション】

クロス集計オプション（crossOptions）は、DB テーブル・ビューのクロス集計を制御するための各種オプションから成るオブジェクトです。このオブジェクトはスクリーンモジュールの【クロス集計】を用いてアプリ利用者が作成します。作成された crossOptions オブジェクトは、クロス集計モジュールの【Update イベント】のハンドラを通じてスクリーンプログラムに渡されます。Update イベントハンドラの作例については【Update イベント】の節を参照してください。

【パタン文字列】

パタン文字列は検索キーワードに用いられるパーセント記号（%）を含んだ文字列です。文字列の最初および最後に現れるパーセント記号（%）は 0 個以上の文字の並びと一致します。これにより前方一致検索、後方一致検索が実現できます。パタン文字列には % の現れる位置によって次の 4 通りがあります。

1. パタン文字列の最初と最後が％のときは部分一致検索が行なわれます。たとえばパタン文字列 "% 人 %" は " 人 "、" 人数 "、" 個人 "、" 百人力 " などと一致します。
2. パタン文字列の最初の文字が％のときは後方一致検索が行なわれます。たとえばパタン文字列 "% 木 " は " 木 "、" 大木 "、" 六本木 " などと一致します。
3. パタン文字列の最後の文字が％のときは前方一致検索が行なわれます。たとえばパタン文字列 " 日本 %" は " 日本 "、" 日本人 "、" 日本列島 " などと一致します。
4. それ以外のときはパタン文字列の最初と最後に％があるものとして 1. の場合と同様に部分一致検索が行なわれます。

【サブクエリ】

サブクエリ（副問い合わせ）は DB テーブル･ビューへの問い合わせに別の問い合わせを埋め込む SQL 文法の仕組みです。Buddy ではサブクエリをオブジェクトで表します。サブクエリオブジェクトを用いることで複数の問い合わせが入れ子になった複雑な問い合わせを記述できます。

サブクエリオブジェクトを作成するには【ModelTable】オブジェクトおよび【ModelView】オブジェクトの select メソッドを用います。以下に例を示します。

```
var tableName = 'shain';
var subquery = this.models[tableName].select(
                  [{expr: 'ID', as: 'shain_ID'}, 'name', 'age'], {});
```

サブクエリオブジェクトは次の 2 つの目的に利用できます。

1. 絞り込み条件
2. 別の DB テーブル・ビューやサブクエリとの結合

サブクエリを用いた絞り込み条件については【サブクエリによる絞り込み】の節を参照してください。

サブクエリオブジェクトは他の DB テーブル･ビューやサブクエリとの結合（join）を生成するために用いることもできます。利用できる結合方法と対応する【ModelTable】オブジェクトおよび【ModelView】オブジェクトのメソッドを以下に示します。

```
内部結合(inner join) … innerJoinメソッド
左外部結合(left outer join) … leftOuterJoinメソッド
右外部結合(right outer join) … rightOuterJoinメソッド
完全外部結合(full outer join) … fullOuterJoinメソッド
交差結合(cross join) … crossJoinメソッド
```

【イベント API】

本節ではイベントを扱う API について説明します。

【イベントハンドラ】

イベントは、実行中のアプリケーションで発生した事象についてスクリーンプログラムが受け取るメッセージ（通知）です。イベントは大きく分けて次の 3 種類があります。

a. マウスボタンの押下やキーボード入力、画面サイズの変更などアプリ利用者の入力デバイスの操作によって発生するイベント（Click,KeyPress,onResizeなど）。
b. スクリーンプログラムの起動や終了、画像ファイルの読み込みエラーなどの事象に応じて発生するイベント（onLoad,onUnload, onErrorなど）。
c. スクリーンプログラムから別のスクリーンプログラムへアプリケーション固有の事象の発生を通知するイベント（onIterateなど）。

あるイベントが発生したという通知をスクリーンプログラムが受け取るには、そのイベントに呼応する名前のイベントハンドラをスクリーンプログラムの var events = { ... から始まるブロックに記述します。たとえば BUTTON1 というモジュール名のボタンが押されたときに生じる Click イベントを受け取るには以下のようなイベントハンドラを追加します。

```
BUTTON1_onClick: function(evt){
    // ここにClickイベントの処理内容を記述します。例:
    api.dialog.alert("BUTTON1が押されました。");
},
```

【イベントオブジェクト】

イベントオブジェクトは、一部のイベントハンドラに evt 引数として渡されてくるオブジェクトです。イベントオブジェクトがハンドラの evt 引数に渡されてくるイベントには以下の 4 種類があります。

```
・キーボードイベント
・フォーカスイベント
・マウスイベント
・画像イベント
```

イベントオブジェクトは以下のプロパティから成ります。

```
target   … イベントの対象となるオブジェクト（スクリーンモジュールなど）
original … イベントを詳細に記述する種々のプロパティから成るオブジェクト
```

original オブジェクトの持つプロパティは、発生したイベントによって異なります。

【キーボードイベント】

キーボードイベントには以下のイベントがあります。

```
onKeyDown
onKeyPress
onKeyUp
```

これらのイベントに対する original オブジェクトは次のデータ型と名前のプロパティから成ります。

```
boolean altKey
number charCode
boolean ctrlKey
boolean getModifierState(key)
string key
number keyCode
string locale
number location
boolean metaKey
boolean repeat
boolean shiftKey
number which
```

【フォーカスイベント】

フォーカスイベントには以下のイベントがあります。

```
onBlur
onFocus
```

これらのイベントに対する original オブジェクトは次のデータ型と名前のプロパティから成ります。

```
DOMEventTarget relatedTarget
```

【マウスイベント】

マウスイベントには以下のイベントがあります。

```
onClick
onContextMenu
onDoubleClick
onMouseDown
onMouseUp
```

これらのイベントに対する original オブジェクトは次のデータ型と名前のプロパティから成ります。

```
boolean altKey
number button
number buttons
number clientX
number clientY
boolean ctrlKey
boolean getModifierState(key)
boolean metaKey
number pageX
number pageY
DOMEventTarget relatedTarget
number screenX
number screenY
boolean shiftKey
```

【画像イベント】

画像イベントには以下のイベントがあります。

```
onError
```

このイベントに対する original オブジェクトは固有のプロパティを持ちません。

【フィルタ API】

フィルタ API は、文字列や日付、日時などのデータを整形して別の文字列に変換するための仕組み（これをフィルタと呼びます）を提供します。

【フィルタ名】

適用するフィルタは名前で指定します。フィルタ名には以下のものがあります。

TRIM　文字列の先頭と末尾の空白文字を取り除く
COMMA　桁区切りのカンマを挿入する
YEN　￥表記にして桁区切りのカンマを挿入する
YMD　日付（Date 型オブジェクトまたは Date 型に変換できる日付文字列）を yyyy/mm/dd の形にする

```
オプション：
-h　　… 区切り文字を / ではなく - にする
-1　　… 1 桁の数字の前に 0 を追加しない
```

JYMD　日付（Date 型オブジェクトまたは Date 型に変換できる日付文字列）を yyyy 年 mm 月 dd 日 の形にする

```
オプション：
-1　　… 1 桁の数字の前に 0 を追加しない
```

GYMD　日付(Date 型オブジェクトまたは Date 型に変換できる日付文字列)を 元号 y 年 m 月 d 日 の形にする
YMDHMS　日時（Date 型オブジェクトまたは Date 型に変換できる日付文字列）を yyyy/mm/dd hh:mm:ss の形にする

```
オプション：
-h　　… 区切り文字を / ではなく - にする
-1　　… 1 桁の数字の前に 0 を追加しない
```

JYMDHMS　日時(Date 型オブジェクトまたは Date 型に変換できる日付文字列)を yyyy 年 mm 月 dd 日 hh 時 mm 分 ss 秒 の形にする

```
オプション：
-1　　… 1 桁の数字の前に 0 を追加しない
```

HMS　時刻(Date 型オブジェクトまたは Date 型に変換できる日付文字列) を hh:mm:ss の形にする

```
オプション：
-1　　… 1 桁の数字の前に 0 を追加しない
```

AGE　年齢を計算する

```
オプション：
-d　　… 年齢を日単位で計算する（誕生日前日に年を取る）
```

PRE　前に指定文字列を追加する

```
オプション：
- 文字列　… 付加する文字列
```

POST　後ろに指定文字列を追加する

オプション：
- 文字列　… 付加する文字列

ROUND　数値を指定桁で丸める

オプション：
- 数字　　… 丸める桁。正の整数なら小数点以下の桁指定、負の整数なら小数点以上の桁指定。指定しなければ小数点以下を四捨五入する
-d　　… 切り捨てる
-u　　… 切り上げる

HIRA2KATA　ひらがなをカタカナに置換する
KATA2HIRA　カタカナをひらがなに置換する
ZEN2HAN　全角英数記号を半角文字に置換する
HAN2ZEN　半角英数記号を全角文字に置換する
ZEN2HANKATA　全角カナを半角カナに置換する
HAN2ZENKATA　半角カナを全角カナに置換する
KYUU2SHIN　旧漢字を新漢字に置換する

【フィルタオプション】

フィルタにはオプションを有するものがあります。オプションを指定するにはフィルタ名のあとにハイフン（-）でオプションをつなぎます。例：

```
AGE-d
```

オプションは複数指定できます。例：

```
YMD-h-1
```

ハイフンが連続したときは先頭のハイフンのみが区切りとなります。例えば ROUND フィルタで丸める桁として -2（十の位）を指定するには次のようにオプションを与えます。

```
ROUND--2
```

【フィルタの連結】

複数のフィルタ名を | で連結すると前段のフィルタの結果が後段のフィルタの入力になります。たとえばフィルタ名として TRIM|COMMA を指定すると、入力データに TRIM フィルタを適用した結果をさらに COMMA フィルタで処理します。

【数式 API】

本節では数式モジュールの API について説明します。

【数式モジュール】

数式モジュールは、他の 1 つ以上のモジュールが保持する現在の値に基づいて計算を実行し、その計算結果を表示するモジュールです。計算処理は他のモジュール（依存関係にあるモジュール）の値が変化したときに自動的に実行されます。実行される計算の内容を記述した関数を数式ハンドラと呼びます。数式ハンドラは、スクリーンプログラムの var formulas = { ... で始まるブロックに記述します。

【数式ハンドラ】

数式ハンドラは、依存関係にある他のモジュールの値が変化したときに数式モジュールの値を更新するために実行される関数です。数式ハンドラの名前は数式モジュールの名前と同じでなければなりません。

たとえば 2 つのデータ入力モジュール DATAINPUT1, DATAINPUT2 に入力されている値を足し合わせて数式モジュール FORMULA1 に表示するには以下のような数式ハンドラを追加します。

```
FORMULA1: function(util){
    return util.sum('DATAINPUT1', 'DATAINPUT2');
},
```

util 引数には数式ハンドラの記述に便利な関数がまとめられた util オブジェクトが渡されてきます。上述の例で用いている util.sum() は引数で与えられた名前の 2 つのモジュールの現在の値を足し合わせた値（総和）を計算する関数です。util オブジェクトの詳細については次の【util オブジェクト】の節を参照してください。

【util オブジェクト】

util オブジェクトは、数式ハンドラに引数として渡されてくる数式ハンドラの記述に便利な関数がまとめられたオブジェクトです。

【util.array.avg】

util.array.avg 関数は arr 引数として数値の配列を受け取り、全要素の平均値を返します。

```
util.array.avg(arr)
```

【util.array.count】

util.array.count 関数は arr 引数として数値の配列を受け取り、配列の要素数を返します。

```
util.array.count(arr)
```

【util.array.max】

util.array.max 関数は arr 引数として数値の配列を受け取り、全要素のうち最大値を返します。

```
util.array.max(arr)
```

【util.array.min】

util.array.min 関数は arr 引数として数値の配列を受け取り、全要素のうち最小値を返します。

```
util.array.min(arr)
```

【util.array.sum】

util.array.sum 関数は arr 引数として数値の配列を受け取り、全要素の総和を返します。

```
util.array.sum(arr)
```

【util.avg】

util.avg 関数は引数で与えられた 1 つ以上の名前のモジュールの現在の値の平均値を返します。

```
util.avg(name1, name2, ...)
```

【util.count】

util.count 関数は引数で与えられた 1 つ以上の名前のモジュールの現在の値の個数（つまり引数の数）を返します。

```
util.count(name1, name2, ...)
```

【util.int】

util.int 関数は name 引数で与えられた名前のモジュールの現在の値を整数値として返します。

```
util.int(name)
```

【util.max】

util.max 関数は引数で与えられた 1 つ以上の名前のモジュールの現在の値のうち最大値を返します。

```
util.max(name1, name2, ...)
```

【util.min】

util.min 関数は引数で与えられた 1 つ以上の名前のモジュールの現在の値のうち最小値を返します。

```
util.min(name1, name2, ...)
```

【util.num】

util.num 関数は name 引数で与えられた名前のモジュールの現在の値を実数値として返します。

```
util.num(name)
```

【util.sum】

util.sum 関数は引数で与えられた 1 つ以上の名前のモジュールの現在の値の総和を返します。

```
util.sum(name1, name2, ...)
```

【サーバー機能】

本節ではサーバー機能設計の JavaScript API について述べます。

【概要】

サーバー機能設計では、サーバー側で実行される JavaScript プログラム（以下、サーバー機能スクリプトと呼びます）を作成します。サーバー機能スクリプトは機能名で識別され、スクリーンプログラムから呼び出されます。スクリーンプログラムはブラウザ（クライアント側）で動作するのに対して、サーバー機能スクリプトはサーバー上で動作します。

【実行タイプ】

サーバー機能には 2 つの実行タイプがあります。

同期実行 … スクリプトの実行が完了した時点の変数resultの内容がクライアント側に返されます。
非同期実行 … スクリプトの中であらかじめ用意されているcallbackという関数を呼び出した時に、その引数（第1引数がエラー、第2引数が結果）によって結果がクライアント側に返されます。

同期実行のサーバー機能スクリプトの例を以下に示します。

```
// 同期実行のサーバー機能スクリプト（機能名mul）
result = options.a * options.b;
```

上と同じ処理を非同期実行するサーバー機能スクリプトの例を以下に示します。

```
// 非同期実行のサーバー機能スクリプト（機能名mul）
var result = options.a * options.b;
callback(null, result);
```

【呼び出し方法】

スクリーンプログラムからサーバー機能を呼び出すには api.serverFunction モジュールを用います（【api.serverFunction】の節を参照）。第 1 引数には機能名、第 2 引数にはオプションオブジェクト、第 3 引数にはコールバック関数を与えます。

オプションオブジェクトはサーバー機能スクリプトに入力データを与えるためのもので、スクリプトの中であらかじめ用意された options という変数に渡されます。

コールバック関数は、スクリプトの実行が完了して結果が返される時に呼ばれます。コールバック関数の第 1 引数はエラーオブジェクト、第 2 引数は結果オブジェクトです。エラーオブジェクトを err とすると、err.response.text プロパティに {"message": エラーメッセージ , "console": {"log": [{"type": "log", "body": ログメッセージ }, ...]}} のような JSON 形式でエラーメッセージと console.log() の出力内容が渡されます。結果オブジェクトは {"result": 結果 , "console": {"log": [...]}} という形式のオブジェクトで、サーバー機能スクリプトの返した結果と console.log() の出力内容が得られます。

前節のサーバー機能スクリプトの例(機能名 mul)を呼び出すスクリーンプログラムの例を以下に示します。

```
api. serverFunction.execute("mul", {a: 2, b: 3}, function(err, res) {
  if( err ) {
    console.log(JSON.parse(err.response.text));
  } else {
    console.log(res.console.log);
    // res.result で得られる結果を用いた処理
  }
});
```

非同期のサーバー機能を executeAsync() で呼び出すとき、呼ばれたサーバー機能側で callback() を 2 度実行するとエラーになります。

【定義済み関数・変数】

サーバー機能のスクリプト内では以下の変数が予め用意されています。

```
tables   … DBテーブルを表すModelTableオブジェクト(【ModelTable】の節を参照)の配列。
views    … DBビューを表すModelViewオブジェクト(【ModelView】の節を参照)の配列。
models   … ModelTableオブジェクトおよびModelViewオブジェクトの配列。
```

また、以下の関数およびライブラリが用意されています。

```
api.sendMessage … スクリーンプログラムのapi.request.sendMessage関数と同等
        (【sendMessage】の節を参照)。
api.serverFunction … サーバー機能から別のサーバー機能を呼び出します(後述)。
api.transaction … DBテーブル・ビューの操作のトランザクション処理を行います(後述)。
api.xml2js … xml2jsモジュール。スクリーンプログラムのapi.lib.xml2jsモジュールと同等
        (【api.lib.xml2js】の節を参照)。
async    … asyncモジュール。スクリーンプログラムのapi.lib.asyncモジュールと同等
        (【api.lib.async】の節を参照)。
console.log(出力文字列) … デバッグ用のメッセージを出力します。出力内容は結果の一部と
        してクライアント側に返されます。
customLog(出力文字列) … カスタムログに文字列を出力します。
lib.BuddyFile … ファイルやディレクトリを操作します(後述)。
lib.BuddyReport … レポート出力をサポートします(後述)。
lib.FileManager … 一時ファイルの管理を担うファイルマネージャを操作します(後述)。
lib.sendMail … サーバを通じてメールを送信します(後述)。
lib.Workbook … Excelファイルの読み書きをサポートします(後述)。
lib.XPDFJ … テキストや画像を含むPDFファイルを作成します(後述)。
```

【利用できるモジュール】

サーバー機能スクリプトの中で require() により利用できるモジュールは以下の通りで

す。

- async
- events
- crypto
- object-assign
- nodemailer
- decimal
- iconv-lite

【api.serverFunction】

サーバー機能から別のサーバー機能を呼び出すには api.serverFunction モジュールを利用します（スクリーンプログラムの【api.serverFunction】モジュールの節を参照）。同期実行には api.serverFunction.execute()、非同期実行には api.serverFunction.executeAsync() を用います。options 引数に与えるオブジェクトには、値が数値か文字列のプロパティのみ指定できます。

サーバー機能の権限チェックはスクリーンプログラムからの呼び出しと同様に行われます。

サーバー機能の再帰的な呼び出しが可能です。ただし再帰呼び出し回数の上限は 10000 回に制限されています。

【api.transaction】

api.transaction 関数は DB テーブル・ビューの操作のトランザクション処理を行うための仕組みを提供します。

新しいトランザクションを開始して、そのトランザクション内でデータベース操作を実行するには、次のように api.transaction 関数を実行します。

```
api.transaction(options, callback)
```

options 引数は今のところ使用されません。callback 引数にはトランザクション内で実行するデータベース操作を記述したコールバック関数を与えます。以下に例を示します。

```
api.transaction(options, function(transaction) {
    // ここにトランザクション内で実行するデータベース操作を記述する
    ...
});
```

コールバック関数の引数には、以下に述べるプロパティおよび関数から成るオブジェクト（これを transaction オブジェクトと呼びます）が渡されます。transaction オブジェクトのプロパティや関数により記述されたデータベース操作はトランザクション内で実行されます。

【プロパティ】

【transaction.tables】

このプロパティの値は、DB テーブルを表す ModelTable オブジェクト（【ModelTable】の節を参照）の配列です。この ModelTable オブジェクトのメソッド呼び出しによる DB テーブルの操作はトランザクション内で実行されます。

注意：【定義済み関数・変数】の節で述べた定義済み変数tablesにより得られるModelTableオブジェクトを通じたDBテーブルの操作は、トランザクション内では実行されません。

【transaction.views】

このプロパティの値は、DB ビューを表す ModelView オブジェクト（【ModelView】の節を参照）の配列です。この ModelView オブジェクトのメソッド呼び出しによる DB ビューの操作はトランザクション内で実行されます。

注意：【定義済み関数・変数】の節で述べた定義済み変数viewsにより得られるModelViewオブジェクトを通じたDBビューの操作は、トランザクション内では実行されません。

【関数】

【transaction.commit】

transaction.commit 関数は、トランザクション内で実行したデータベース操作をコミット（確定）してトランザクションを終了します。

```
transaction.commit(callback)
```

callback 引数にはコールバック関数を与えます。コミットが失敗するとコールバック関数の引数には Error オブジェクトが渡され、成功すると null が渡されます。

【transaction.rollback】

transaction.rollback 関数は、トランザクション内で実行したデータベース操作をロールバック（破棄）してトランザクションを終了します。

```
transaction.rollback(callback)
```

callback 引数にはコールバック関数を与えます。ロールバックが失敗するとコールバック関数の引数には Error オブジェクトが渡され、成功すると null が渡されます。

【lib.BuddyFile】

lib.BuddyFileクラスはアプリケーションのfilesディレクトリ配下のファイルおよびディレクトリを操作するための仕組みを提供します。

【コンストラクタ】

BuddyFile オブジェクトを作成するには JavaScript の new 演算子を用います。

```
new lib.BuddyFile(filePath, options)
```

引数：

```
filePath … ファイルまたはディレクトリへのパス名。
options  … オプション。
```

filePath 引数には操作対象となるファイルまたはディレクトリをアプリケーションの files ディレクトリからの相対パス名で与えます。options 引数には以下のプロパティから成るオブジェクトを与えます。

```
temporary … trueを与えると一時ファイルを作成します。filePath引数に与えたパス名は無視
            され、一時ファイル名が自動的に与えられます。作成された一時ファイルはサーバー機
            能スクリプトの実行完了時に自動的に削除されます。
```

【メソッド】

【append】

append メソッドはデータをファイルに追加します。ファイルが存在しない場合は新しく作成されます。

```
append(data, options, callback)
```

data 引数にはファイルに書き込むデータを文字列で与えます。options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode   … data引数に与えた文字列の文字コード(省略可)。
```

callback 引数にはコールバック関数を与えます。書き込みが成功したらコールバック関数の引数には null、失敗したら Error オブジェクトが渡されます。

【appendChild】

appendChild メソッドは BuddyFile オブジェクトのパス名からの相対パスを表す新しい BuddyFile オブジェクトを返します。

```
appendChild(name)
```

name 引数には BuddyFile オブジェクトのパス名からの相対パスを文字列で与えます。

以下に例を示します。

```
var importDir = new BuddyFile('import', {});
var importFile = importDir.appendChild('table1.csv');
```

このとき変数 importFile の値は import/table1.csv を表す BuddyFile オブジェクトです。

【basename】

basename メソッドは BuddyFile オブジェクトのパス名のうちファイル名を返します。

```
basename(ext)
```

ext 引数には拡張子を文字列で与えます（省略可）。ファイル名の拡張子と一致した場合には拡張子のないファイル名が返されます。

【createReadStream】

createReadStream メソッドはファイルからの読み込みストリームを作成します。

```
createReadStream(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode    … 読み込みストリームの文字コード（省略可）。
parse     … ファイル形式（省略可）。文字列"csv"を与えるとファイルをCSV形式と見なして解析
          します。
```

【createWriteStream】

createWriteStream メソッドはファイルへの書き込みストリームを作成します。

```
createWriteStream(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode    … 書き込みストリームの文字コード。
```

【extname】

extname メソッドは BuddyFile オブジェクトのパス名のうちファイル名の拡張子を返します。

```
extname()
```

【getFilePath】

getFilePath メソッドは BuddyFile オブジェクトのパス名を返します。

```
getFilePath()
```

【mkdir】

mkdir メソッドはディレクトリを作成します。

```
mkdir(options, callback)
```

options 引数は今のところ使用しません。callback 引数にはコールバック関数を与えます。ディレクトリの作成に成功したらコールバック関数の引数には null、失敗したら Error オブジェクトが渡されます。

【read】

read メソッドはファイルからデータを読み込みます。

```
read(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode    … ファイルの文字コード(省略可)。
parse     … ファイル形式(省略可)。文字列"csv"を与えるとファイルをCSV形式と見なして解析
            します。文字列"xlsx"を与えるとXLSXファイルとして開き、Workbookオブジェクトを返
            します(【lib.Workbook】の節を参照)。
```

callback 引数にはコールバック関数を与えます。データの読み込みに成功したらコールバック関数の第 1 引数には null、第 2 引数には読み込み結果が渡されます。失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【readdir】

readdir メソッドはディレクトリ内のファイルおよびサブディレクトリの一覧を返します。

```
readdir(options, callback)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode    … ファイル名の文字コード(省略可)。
```

callback 引数にはコールバック関数を与えます。一覧の取得に成功したらコールバック関数の第 1 引数には null、第 2 引数にはファイルおよびサブディレクトリを表す BuddyFile オブジェクトの配列が渡されます。失敗したらコールバック関数の第 1 引数に Error オブジェクトが渡されます。

【remove】

remove メソッドはファイルまたはディレクトリを削除します。

```
remove(options, callback)
```

options 引数は今のところ使用しません。callback 引数にはコールバック関数を与えます。削除に成功したらコールバック関数の引数には null、失敗したら Error オブジェクトが渡されます。

【stat】

stat メソッドはパスの状態を取得します。

```
stat(options, callback)
```

options 引数は今のところ使用しません。callback 引数にはコールバック関数を与えます。状態の取得に成功したらコールバック関数の第 1 引数には null、第 2 引数にはパスの状態を表すオブジェクト（Node.js の fs.Stats オブジェクト）が渡されます。状態の取得に失敗した場合はコールバック関数の第 1 引数に Error オブジェクトが渡されます。

fs.Stats オブジェクトはサーバーの実行環境によって詳細が異なります。主なプロパティについて以下に示します。

```
atime    … 最終アクセス日時(Dateオブジェクト)
ctime    … 最終状態変更日時(Dateオブジェクト)
mtime    … 最終修正日時(Dateオブジェクト)
size     … ファイルサイズ(バイト数)
```

fs.Stats オブジェクトの例を以下に示します。

```
{
    dev: 64768,
    mode: 16895,
    nlink: 2,
    uid: 0,
    gid: 0,
    rdev: 0,
    blksize: 4096,
    ino: 3038506,
    size: 4096,
    blocks: 8,
    atime: "2017-10-10T14:25:58.910Z",
    mtime: "2017-10-10T14:25:58.909Z",
    ctime: "2017-10-10T14:25:58.909Z",
    birthtime: "2017-10-10T14:25:58.909Z"
}
```

fs.Stats オブジェクトの詳細については下記 URL の Node.js ドキュメントを参照してください。

```
https://nodejs.org/api/fs.html#fs_class_fs_stats
```

【unzip】

unzip メソッドは ZIP ファイルの内容を展開します。

```
unzip(unzipDir, options, callback)
```

unzipDir 引数には展開先のディレクトリを表す BuddyFile オブジェクトを与えます。

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode    … ファイル名の文字コード(省略可)。省略した場合は文字コードを自動検出します。
            文字コードを変更しない場合は"through"を指定します。
```

【write】

write メソッドはデータをファイルに書き込みます。ファイルが存在しない場合は新しく作成されます。

```
write(data, options, callback)
```

data 引数にはファイルに書き込むデータを文字列で与えます。options 引数には以下のプロパティから成るオブジェクトを与えます。

```
encode    … data引数に与えた文字列の文字コード(省略可)。
```

callback 引数にはコールバック関数を与えます。書き込みが成功したらコールバック関数の引数には null、失敗したら Error オブジェクトが渡されます。

【zip】

zip メソッドは BuddyFile オブジェクトが表すファイルまたはディレクトリを ZIP ファイルに追加します。ZIP ファイルが存在しなければ新たに作成します。

```
zip(zipFile, options, callback)
```

zipFile 引数には ZIP ファイルを表す BuddyFile オブジェクトを与えます。options 引数は今のところ使用しません。ZIP ファイルへの書き込みに成功したらコールバック関数の引数には null、失敗したら Error オブジェクトが渡されます。

【lib.BuddyReport】

lib.BuddyReport モジュールは BuddyFile オブジェクトを出力先としてレポートを出力する API を提供するモジュールです。

【関数】

【output】

output 関数は BuddyFile オブジェクトを出力先としてレポートを出力します。

```
output(name, file, options, callback)
```

name 引数にはレポート名を与えます。file 引数にはレポートの出力先となる BuddyFile オブジェクトを与えます。options 引数には以下のプロパティから成るオブジェクトを与えます。

```
where    … 絞り込み条件(【絞り込み条件】の節を参照)。
order    … 並べ替え指定(【並べ替え】の節を参照)。
offset   … 最初のレコード番号(先頭が0)。
num      … 出力件数。
param    … パラメーターオブジェクト
```

param プロパティには任意のプロパティから成るオブジェクトを指定できます。ここで与えたプロパティの値はレポート中で「this.param. プロパティ名」で参照できます。callback 引数にはコールバック関数を与えます。レポート出力が成功したらコールバック関数の引数には null、失敗したら Error オブジェクトが渡されます。

【lib.FileManager】

lib.FileManager モジュールは、サーバー機能スクリプトからファイルマネージャを利用するための API を提供するモジュールです。アプリケーションの files ディレクトリ配下のファイルは固有の URL を持ち、ファイルが存在する間は常にダウンロード可能です。一方、ファイルマネージャは有効期限のある URL をファイルに与えて一時的にダウンロードを許す仕組みを提供します。有効期限が過ぎるとファイルは自動的に削除され、ファイルに付与された URL は無効になります。ファイルマネージャの管理下にあるファイルを「一時ファイル」と呼びます。

【関数】

【getURL】

getURL 関数は一時ファイルの識別子（ハッシュ）からそのファイルをダウンロードできる URL を得ます。

```
getURL(hash)
```

hash 引数には一時ファイルの識別子となるハッシュ値を文字列で与えます。

【register】

register 関数は BuddyFile オブジェクトが表すファイルをファイルマネージャに登録します。

```
register(file, options, callback)
```

file 引数には登録対象となるファイルを表す BuddyFile オブジェクトを与えます。options 引数には以下のプロパティから成るオブジェクトを与えます。

```
auth     … 権限情報オブジェクト(省略可)。省略すると同じアプリケーションからのみアクセスを許可します。
fileName … ファイル名(省略可)。省略すると元のファイル名を用います。
limit    … 有効期限(省略可)。ファイルを削除する日時をDateオブジェクトで与えます。省略すると登録から1時間有効となります。
```

callback 引数にはコールバック関数を与えます。ファイルの登録に失敗するとコールバック関数の第 1 引数には Error オブジェクトが渡されます。登録に成功した場合、コールバック関数の第 1 引数には null、第 2 引数にはハッシュ値が文字列で渡されます。ここで得られたハッシュ値は上述の getURL 関数でファイルのダウンロード URL を得るために用います。

【lib.sendMail】

lib.sendMail 関数はサーバを通じてメールを送信します。

```
lib.sendMail(maildata, callback)
```

maildata 引数には以下のプロパティから成るオブジェクトを与えます。

```
type      … メール本文の形式。"text"または"html"
from      … 送信元メールアドレス（省略するとBuddyサーバの設定ファイルで指定された送信
          元メールアドレスを使用）
to        … 宛先メールアドレス（カンマ区切りで複数指定可）
cc        … CCメールアドレス（カンマ区切りで複数指定可）
bcc       … BCCメールアドレス（カンマ区切りで複数指定可）
subject   … 表題
body      … 本文
attachments … 添付ファイルパス（filesディレクトリからの相対パス）の配列
smtpserver … SMTPサーバのホスト名またはIPアドレス（省略するとBuddyサーバの設定ファイ
          ルで指定されたSMTPサーバを使用）
```

callback 引数にはコールバック関数を与えます。メールの送信に失敗した場合にはコールバック関数の第 1 引数に Error オブジェクトが渡されます。メールの送信に成功した場合は、第 1 引数には null、第 2 引数には以下のプロパティから成る SMTP 応答オブジェクトが渡されます。

```
envelope … エンベロープオブジェクト
accepted … SMTPサーバによって受理されたメールアドレスの配列
rejected … SMTPサーバによって拒否されたメールアドレスの配列
response … SMTPサーバからの応答メッセージ文字列
```

少なくとも 1 つのメールアドレスが SMTP サーバにより受理されたら送信成功として扱われます。

【lib.Workbook】

lib.Workbook クラスは Excel ワークブックを読み書きするための API を提供するクラスです。

Workbook クラスは Node.js の exceljs モジュールに基づいています。exceljs の詳細については下記 URL のドキュメントを参照してください。

```
https://www.npmjs.com/package/exceljs
```

【ワークブック】

Workbook オブジェクトを作成するには JavaScript の new 演算子を用います。

```
var workbook = new lib.Workbook();
```

既存の XLSX ファイルを読み込んで Workbook オブジェクトを作成するには、次の例のように BuddyFile オブジェクトの read() メソッドの第 1 引数に {parse: "xlsx"} オプションを与えます。

```
var xlsx = new BuddyFile("import/test.xlsx");
xlsx.read({parse: "xlsx"}, function(err, workbook){
  // ここでworkbookはWorkbookオブジェクト
});
```

Workbook オブジェクトを XLSX 形式のファイルとして保存するには、次の例のように BuddyFile オブジェクトの write() メソッドの第 1 引数に Workbook オブジェクトを与えます。

```
var xlsx = new BuddyFile("output/test.xlsx");
xlsx.write(workbook, function(err){
  // ...
});
```

【ワークシート】

ワークブックに新しいワークシートを追加するには次のようにします。

```
var sheet = workbook.addWorksheet('My Sheet');
```

ワークブックからワークシートを得るには次の例のように getWorksheet() を用います。引数には名前または番号（先頭が 1）を与えます。

```
// 名前で探す
var worksheet = workbook.getWorksheet('My Sheet');
// 番号で探す
var worksheet = workbook.getWorksheet(1);
```

【カラム】

ワークシートのカラムにカラム名（key)、表示名（header)、幅（width）などを設定するには次の例のようにワークシートの columns プロパティを変更します。

```
worksheet.columns = [
  { key: 'id', header: 'ID', width: 10 },
  { key: 'name', header: '名前', width: 30 },
  { key: 'dob', header: '生年月日', width: 20 }
];
```

ワークシートからカラムを得るには次の例のように getColumn() を用います。引数にはカラム名、英文字、番号（先頭が 1）を与えます。

```
var idCol = worksheet.getColumn('id');
var nameCol = worksheet.getColumn('B');
var dobCol = worksheet.getColumn(3);
```

カラムのプロパティを変えるには次のようにします。

```
nameCol.key = 'name';
nameCol.header = '氏名';
nameCol.width = 15;
```

カラムの各セルについて同じ処理を繰り返し実行するには次の例のように eachCell() を用います。

```
nameCol.eachCell(function(cell, rowNumber) {
  // cellはセルオブジェクト、rowNumberは行番号
});
```

【行】

ワークシートに新しい行を追加するには次の例のように addRow() を用います。引数にはカラム名をプロパティとするオブジェクトを与えます。ここで指定するカラム名（id、name など）は、前節に示した方法で予め定義しておく必要があります。

```
worksheet.addRow({id: 1, name: '近藤勇', dob: new Date(1834,11,9)});
worksheet.addRow({id: 2, name: '土方歳三', dob: new Date(1835,5,31)});
```

addRow() の引数には配列を与えることもできます。この場合、値はカラム A から順に格納されます。

```
worksheet.addRow([3, '斎藤一', new Date(1844,2,18)]);
```

ワークシートから行を得るには次の例のように getRow() を用います。引数には番号（最初が 1）を与えます。

```
var row = worksheet.getRow(5);
```

ワークシートの最後の編集可能な行を得るには次の例のように lastRow プロパティを用います（編集可能な行がないときは undefined）。

```
var row = worksheet.lastRow;
```

行の高さを設定するには次の例のように height プロパティを変更します。

```
row.height = 42.5;
```

行の値を一括して変更するには次の例のように values プロパティを用います。

```
row.values = { id: 3, name: '斎藤一', dob: new Date(1844,2,18) };
```

行のセルを得るには次の例のように getCell() を用いて名前、英文字、番号（先頭が 1）を与えます。

```
row.getCell('id').value = 3;
row.getCell('B').value = '斎藤一';
row.getCell(3).value = new Date(1844,2,18);
```

ワークシートの各行について同じ処理を繰り返し実行するには次の例のように eachRow() を用います。

```
worksheet.eachRow(function(row, rowNumber) {
  // rowは行オブジェクト、rowNumberは行番号
  console.log('Row ' + rowNumber + ' = ' + JSON.stringify(row.values));
});
```

【セル】

ワークシートからセルを得るには次の例のように getCell() を用います。

```
worksheet.getCell('C3').value = new Date(1844,2,18);
```

セルを結合するには次の例のように mergeCells() を用います。

```
worksheet.mergeCells('A4:B5');
```

セルの結合を解除するには次の例のように unMergeCells() を用います。

```
worksheet.unMergeCells('A4');
```

【スタイル】

本節ではセルのスタイルを変更するためのプロパティについて述べます。

【書式設定】

セルの書式設定を変更するには次のように numFmt プロパティを設定します。

```
worksheet.getCell('A1').numFmt = '0.00%';
```

【フォント】

セルのフォントを変更するには次のように font プロパティを設定します。

```
worksheet.getCell('A1').font = { name: 'Arial', size: 12 };
```

セルの font プロパティには次のプロパティから成るオブジェクトを与えます。

```
name      … フォント名('Arial'など)
color     … 色。例えば {argb: 'FFFF0000'} のようにargbプロパティから成るオブジェクトを
            与える。
bold      … 太字(trueまたはfalse)
italic    … 斜体(trueまたはfalse)
underline … 下線(true, false, 'none', 'single', 'double', 'singleAccounting',
            'doubleAccounting'のいずれか)
strike    … 取り消し線(trueまたはfalse)
```

【配置】

セルの配置を変更するには次のように alignment プロパティを設定します。

```
worksheet.getCell('A1').alignment = { vertical: 'top', horizontal: 'left' };
```

セルの alignment プロパティには次のプロパティから成るオブジェクトを与えます。

```
horizontal … 横位置('left', 'center', 'right', 'fill', 'justify',
             'centerContinuous', 'distributed'のいずれか)
vertical … 縦位置('top', 'middle', 'bottom', 'distributed', 'justify'のいずれか)
wrapText … 折り返し(trueまたはfalse)
indent   … 字下げ(整数値)
readingOrder … 文字の方向('rtl'または'ltr')
textRotation … 回転角度(-90から90までの数値、または'vertical')
```

【枠線】

セルの枠線を変更するには次のように border プロパティを設定します。

```
worksheet.getCell('A1').border = {
  top: {style: 'thin'},
  bottom: {style: 'thin'},
  left: {style: 'thin'},
  right: {style: 'thin'}
};
```

セルの border プロパティには次のプロパティから成るオブジェクトを与えます。

```
top       … 上の枠線スタイル
```

```
bottom   … 下の枠線スタイル
left     … 左の枠線スタイル
right    … 右の枠線スタイル
```

枠線スタイルには style プロパティから成るオブジェクトを与えます。style プロパティの有効な値は次の通りです。

```
style    … 'dashDot', 'dashDotDot', 'dotted', 'double', 'hair', 'medium',
           'mediumDashDot', 'mediumDashDotDot', 'mediumDashed', 'slantDashDot',
           'thick', 'thin'のいずれか。
```

【塗りつぶし】

セルの塗りつぶしを変更するには次の例のように fill プロパティを設定します。

```
// パターン(単色)
worksheet.getCell('A1').fill = {
  type: 'pattern',
  pattern: 'solid',
  fgColor: {argb: 'FFFFFF00'}
};
// パターン
worksheet.getCell('A2').fill = {
  type: 'pattern',
  pattern: 'darkHorizontal',
  fgColor: {argb: 'FFFFFF00'},
  bgColor: {argb: 'FF000000'}
};
// グラデーション
worksheet.getCell('A3').fill = {
  type: 'gradient',
  gradient: 'angle',
  degree: 0,
  stops: [
    {position: 0.0, color: {argb: 'FFFF0000'}},
    {position: 0.5, color: {argb: 'FFFFFF00'}},
    {position: 1.0, color: {argb: 'FF00FF00'}}
  ]
};
```

パターンで塗りつぶすには、セルの fill プロパティに以下のプロパティから成るオブジェクトを与えます。

```
type     … 文字列'pattern'(必須)。
pattern  … パターン名(必須)。'darkDown', 'darkGray', 'darkGrid', 'darkHorizontal',
           'darkTrellis', 'darkUp', 'darkVertical', 'darkVertical', 'gray0625',
           'gray125', 'lightDown', 'lightGray', 'lightGrid', 'lightGrid',
           'lightHorizontal', 'lightTrellis', 'lightUp', 'lightVertical', 'mediumGray',
```

```
          'none', 'solid'のいずれか。
fgColor   … 前景色(省略可)。デフォルトは'black'。
bgColor   … 背景色(省略可)。デフォルトは'white'。
```

グラデーションで塗りつぶすには、セルの fill プロパティに以下のプロパティから成るオブジェクトを与えます。

```
type      … 文字列'gradient'(必須)。
gradient … グラデーションタイプ(必須)。'angle'または'path'。
degree    … グラデーション方向の角度('angle'タイプの場合に必須)。0なら左から右へのグラデーション、1から359の値なら時計回りにグラデーション方向を回転。
center    … パスの始点の相対座標('path'タイプの場合に必須)。0から1までの相対値でパスの始点を指定。
stops     … グラデーションを構成する色の配列(必須)。配列の要素は次のプロパティから成るオブジェクト。
            position … 位置。始点を0、終点を1とする相対位置で指定。
            color    … 色。
```

【lib.XPDFJ】

lib.XPDFJ クラスは、テキストや画像を含む PDF ファイルをサーバー機能のスクリプトで生成するための仕組みを提供するクラスです。本クラスの機能の概要は次の通りです。

・サーバー機能のスクリプトでテキスト、段落、直線、矩形、画像をページに配置することで PDF を生成できます。内部的に Perl モジュール XPDFJ を利用しています。

・new lib.XPDFJ() でオブジェクトを作り、そのメソッドでテキスト、段落、直線、矩形、画像を配置していくと、ソースとなるオブジェクトが作られます。このオブジェクトが output() メソッドで XML 形式に変換されて XPDFJ モジュールで処理され、ファイルに出力されます。

・用意されているメソッドでは XPDFJ モジュールやその下の PDFJ ライブラリの全ての機能が使えるものではなく、Buddy のレポート出力で必要になると思われる機能に限定しています。

・テキストと段落について、実際の出力はせずに、それぞれの縦横のサイズを得る方法を用意しています。これによって内容が可変のテキストについて適切な配置が可能となります。

・大きさや位置は単位を指定しなければポイントですが、単位「mm」を付加することでミリメートル指定も可能です（72 ポイント＝ 1 インチ＝ 25.4mm）。

・ソースとなるオブジェクトまたは XML テキストを自分で組み立てておいて渡すようにすれば、XPDFJ や PDFJ の全ての機能を利用できます。XPDFJ のマクロ機能も利用できます。

【コンストラクタ】

XPDFJ オブジェクトを作成するには JavaScript の new 演算子を用います。

```
new lib.XPDFJ(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
src     … ソースオブジェクトまたはXMLテキスト（省略可）
ydir    … y軸（上下方向）をどちら向きとするか（"up"または"down"、省略すると"up"）
```

ソースオブジェクトの作成方法は後述します。src を省略した場合、ソースは後述の XPDFJ クラスのメソッドで組み立てます。ydir による指定は後述のメソッドにおいて y（縦位置指定）と h（図形の縦送り）に影響します（src によるソースには影響しません）。

【メソッド】

本節では XPDFJ クラスのメソッドについて述べます。各メソッドの呼び出しは XPDFJ オブジェクト自身を返すので、次のように書くことが可能です。

```
var xpdfj = new lib.XPDFJ(…);
xpdfj.newDoc(…).addText(…).addLine(…).…
```

【addBox】

addBox メソッドは矩形を追加します。

```
addBox(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
w         … 横送り(右へ)
h         … 縦送り(ydirがupなら上へ、downなら下へ)
linewidth … 線幅(省略すると0.25)
color     … 線色("#RRGGBB"で指定、省略すると黒(#000000))
bgcolor   … 塗りつぶし色("#RRGGBB"で指定、省略すると塗りつぶさず枠線のみ)
x         … 横位置(省略するとページ左端)
y         … 縦位置(省略するとページ上端)
align     … 配置基準(【addText】メソッドの節を参照)
```

ある位置と、そこから w、h で指定した長さだけ移動した位置とを対角とする矩形を描きます。

【addImage】

addImage メソッドは画像を追加します。今のところ PNG 画像のみ利用できます。

```
addImage(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
src       … 画像ファイルをfilesからのパス名で指定
w         … 幅
h         … 高さ
x         … 横位置(省略するとページ左端)
y         … 縦位置(省略するとページ上端)
align     … 配置基準(【addText】メソッドの節を参照)
```

【addLine】

addLine メソッドは直線を追加します。

```
addLine(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
w         … 横送り(右へ)
h         … 縦送り(ydirがupなら上へ、downなら下へ)
linewidth … 線幅(省略すると0.25)
color     … 線色("#RRGGBB"で指定、省略すると黒(#000000))
```

```
x        … 横位置(省略するとページ左端)
y        … 縦位置(省略するとページ上端)
align    … 配置基準(【addText】メソッドの節を参照)
```

ある位置から w､h で指定した長さだけ移動した位置まで直線を引きます。h が 0 なら水平線、w が 0 なら垂直線となります。x、y、align は、w、h の指定によって構成される矩形がどの位置に配置されるかを決めます。オプション指定の組み合わせによって、x、y で指定した位置が直線の端になるとは限らない点に注意してください。

【addParagraph】

addParagraph メソッドは段落を追加します。

```
addParagraph(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
text     … 表示する文字列
font     … フォント("mincho"または"gothic"､省略すると"mincho")
fontsize … フォントサイズ(省略すると10ポイント)
color    … 文字色("#RRGGBB" で指定､省略すると黒(#000000))
size     … 段落の横幅(省略すると100)
textalign … 各行の配置方法(b: 行頭揃え､m: 中央揃え､e: 行末揃え､w: 両端揃え､W: 強制両
        端揃え､省略するとw)｡wによる両端揃えは末尾行だけは行頭揃えとなります｡Wによる強
        制両端揃えは末尾行も含めて両端揃えとなります｡
linefeed … 行送り(省略すると150%)｡値は数値かパーセントで指定します｡パーセント指定す
        るとfontsizeを元に計算します｡linefeedは行間ではなく行送りであることに注意して
        ください｡
x        … 横位置(省略するとページ左端)
y        … 縦位置(省略するとページ上端)
align    … 配置基準(【addText】メソッドの節を参照)
```

text オプションの文字列は size で指定された横幅に収まるように折り返されて表示されます。また、文字列に \n を含めるとそこで折り返されます。

【addText】

addText メソッドはテキストを追加します。

```
addText(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
text     … 表示する文字列
font     … フォント("mincho"または"gothic"､省略すると"mincho")
fontsize … フォントサイズ(省略すると10ポイント)
color    … 文字色("#RRGGBB"で指定､省略すると黒(#000000))
x        … 横位置(省略するとページ左端)
y        … 縦位置(省略するとページ上端)
align    … 配置基準(後述)
```

textオプションの文字列は\rや\nを含んでいても折り返されません。alignによる配置基準は、x、yで指定した位置が表示するもののどこにあたるかを、縦位置については t、m、b のいずれか、横位置については l、c、r のいずれかを組み合わせて指定します。

```
t       … y位置が表示物の上端
m       … y位置が表示物の中央
b       … y位置が表示物の下端
l       … x位置が表示物の左端
c       … x位置が表示物の中央
r       … x位置が表示物の右端
```

align を省略すると、tl（配置基準を表示物の左上）が用いられます。

【getTextSizes】

getTextSizes メソッドはソースに含まれる各テキストと段落のサイズを返します。このメソッドはソースが XPDFJ オブジェクトのメソッドで組み立てられた場合に有効です。

```
getTextSizes(options, callback)
```

options 引数にはオプションオブジェクトを与えますが今のところ使用されません。callback 引数にはコールバック関数を与えます。エラーがあればコールバック関数の第1引数には Error オブジェクトが渡されます。エラーが無ければ、コールバック関数の第1引数には null、第2引数には結果が [横サイズ , 縦サイズ] の配列（単位はポイント）で渡されます。

【newDoc】

newDoc メソッドはドキュメントを作成（ページも一つ自動的に追加）します。

```
newDoc(options)
```

options 引数には以下のプロパティから成るオブジェクトを与えます。

```
pagesize … ページサイズ:"ハガキ"、"ハガキL"、"B5"、"B5L"、"B4"、"B4L"、"A4"、"A4L"、"A3"、"A3
        L"のいずれか(「L」と付くのは横、付かないのは縦)
width    … ページ横幅
height   … ページ高さ
```

pagesize も width、height も指定しなかった場合は A4 縦サイズとなります。

【newPage】

newPage メソッドは新しいページを追加します。以後のページの構成要素はそのページに配置されます。

```
newPage()
```

【ouput】

ouput メソッドは、ソースにもとづいて PDF ファイルを生成します。

```
ouput(file, options, callback)
```

file 引数には出力先ファイルとなる BuddyFile オブジェクトを指定します。options 引数にはオプションオブジェクトを与えますが今のところ使用されません。callback 引数にはコールバック関数を与えます。ファイル出力に敗するとコールバック関数の第 1 引数にはError オブジェクトが渡されます。ファイル出力に成功した場合、コールバック関数の第 1 引数には null、第 2 引数には出力されたファイルのパスが渡されます。XPDFJ クラスのメソッドでソースを組み立てる場合は、その最後に output() を呼び出してください。

【使用例】

【実例 1】

ソースを XPDFJ クラスのメソッドで組み立てる例を以下に示します。

```
var xpdfj = new lib.XPDFJ({ydir: "down"});

xpdfj.newDoc({pagesize: "A4"});
xpdfj.addText({text: "テスト / ntest", x: 50, y: 50, font: "gothic", fontsize: 20,
 color: "#ff0000"});
xpdfj.addParagraph({text: "テスト2 <testtest> なんとか / nかんとか", x: 50, y: 80});
xpdfj.addLine({x: 50, y: 150, w: 200, h: 0});
xpdfj.addBox({x: 50, y: 200, w: 150, h: 200, bgcolor: '#00ffff'});
xpdfj.addImage({src: 'images/kamogawa.png', x: 50, y: 300, w: 327, h: 240});

var bf = new lib.BuddyFile("", {temporary: true});
xpdfj.output(bf, {}, function(err, result){ … });
```

【実例 2】

ソースをオブジェクトで与える例を以下に示します。

```
var src = {XPDFJ: {version: "0.1"}, _:[
   {Doc: {version:"1.3", pagewidth:"595", pageheight:"842"}},
   {new_font: {setvar:"$font", basefont:"Ryumin-Light", encoding:"UniJIS-UCS2
   -HW-H"}},
   {new_page: {setvar:"$page"}},
   {TStyle: {setvar:"$tstyle", font:"$font", fontsize:"20"}},
   {Text: {setvar:"$text", style:"$tstyle", texts:"テスト"}, _:[
       {call: [
           {show: {page:"$page", x:"40", y:"800", align:"tl"}},
       ]},
   ]},
   {print: {file:"$Args{outfile}"}},
]};
```

```
var xpdfj = new lib.XPDFJ({src: src});
var bf = new lib.BuddyFile("", {temporary: true});
xpdfj.output(bf, {}, function(err, result){ … });
```

【ソースオブジェクトの作成】

ソースをオブジェクトで指定する場合は、次のいずれかの形式を組み合わせて入れ子にすることで組み立てます。二番目の形式は属性がない場合に用います。

```
{タグ名: {属性名: 属性値,…}, _:[子オブジェクトまたは文字列,…]}
{タグ名: [子オブジェクトまたは文字列,…]}
```

例えば、前掲の【実例 2】のソースオブジェクトは、次の XML テキストに変換されて、XPDFJ で処理されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<XPDFJ version="0.1">
   <Doc version="1.3" pagewidth="595" pageheight="842"/>
   <new_font setvar="$font" basefont="Ryumin-Light" encoding="UniJIS-UCS2-HW-H"/>
   <new_page setvar="$page"/>
   <TStyle setvar="$tstyle" font="$font" fontsize="20"/>
   <Text setvar="$text" style="$tstyle" texts="テスト">
       <call>
           <show page="$page" x="40" y="800" align="tl"/>
       </call>
   </Text>
   <print file="$Args{outfile}"/>
</XPDFJ>
```

【マクロの利用】

XPDFJ には独自のタグを定義したりできるマクロ機能があります。ソース中で直接マクロ機能を利用することもできますし、マクロファイルを用意して <do file="…"/> で読み込むことで利用することもできます。

lib.XPDFJ でも、files/import/ にマクロファイルを入れておけば、上記のオブジェクト形式なら {do: {file: "…"}} として読み込まれます。

例えば XPDFJ の標準マクロ stddefs.inc を利用すると、次のように HTML ライクなタグを使って書けます（files/import/ に、stddefs.inc、stdfontsH.inc、stdfontsV.inc を入れておく必要があります）。

```
var src = {XPDFJ: {version: "0.2"}, _:[
   {do: {file: "stddefs.inc"}},
   {BODY: [
       {H2: ["見出し"]},
```

```
        {P: ["XPDFJ / r / nテスト"]},
    ]},
    {print: {file:"$Args{outfile}"}},
]};
```

あるいは XML テキストとしてソースを与えるなら次のように記述します。

```
<?xml version="1.0"?>
<XPDFJ version="0.2">
    <do file="stddefs.inc"/>
    <BODY>
        <H2>見出し</H2>
        <P>XPDFJ
        テスト</P>
    </BODY>
    <print file="$Args{outfile}"/>
</XPDFJ>
```

標準マクロの解説は別文書「XPDFJ 標準マクロ概説」を参照してください。