IM-FormaDesigner
Ver.7.2

プログラミングガイド

2012/10/26 第 2 版

＜＜　変更履歴　＞＞

変更年月日	変更内容
2012/03/30	初版
2012/10/26	第 2 版 PDF 出力処理プログラムについて追記

<< 目次 >>

1 はじめに

1.1 目的

本ドキュメントでは、IM-FormaDesigner で利用可能なユーザプログラムの開発方法について解説します。

概要は以下の通りです。

概要	ユーザプログラムについての概要を説明します。 この章を読んだ上で、以降の必要な章をお読みになることをお勧めします。
入力チェック	入力チェックプログラムを実装する方法について説明します。
後処理	後処理プログラムを実装する方法について説明します。
登録データ管理 API	APIを使用してアプリケーションの登録データを取得・操作する方法について説明します。
テンプレートの格納場所	ユーザプログラムのテンプレートの格納場所について説明します。

1.2 前提条件

本書に記述されているサンプルプログラムは、JavaEE 開発モデルおよびスクリプト開発モデルで記述されています。そのため、JavaEE 開発モデルおよびスクリプト開発モデルに関する理解は必須です。各開発モデルに関しては、付属する各種マニュアルおよび API リストを参照してください。

2 概要

IM-FormaDesigner では、作成したアプリケーションに以下のユーザプログラムを組み込むことができます。

- 入力チェックプログラム
- 後処理プログラム

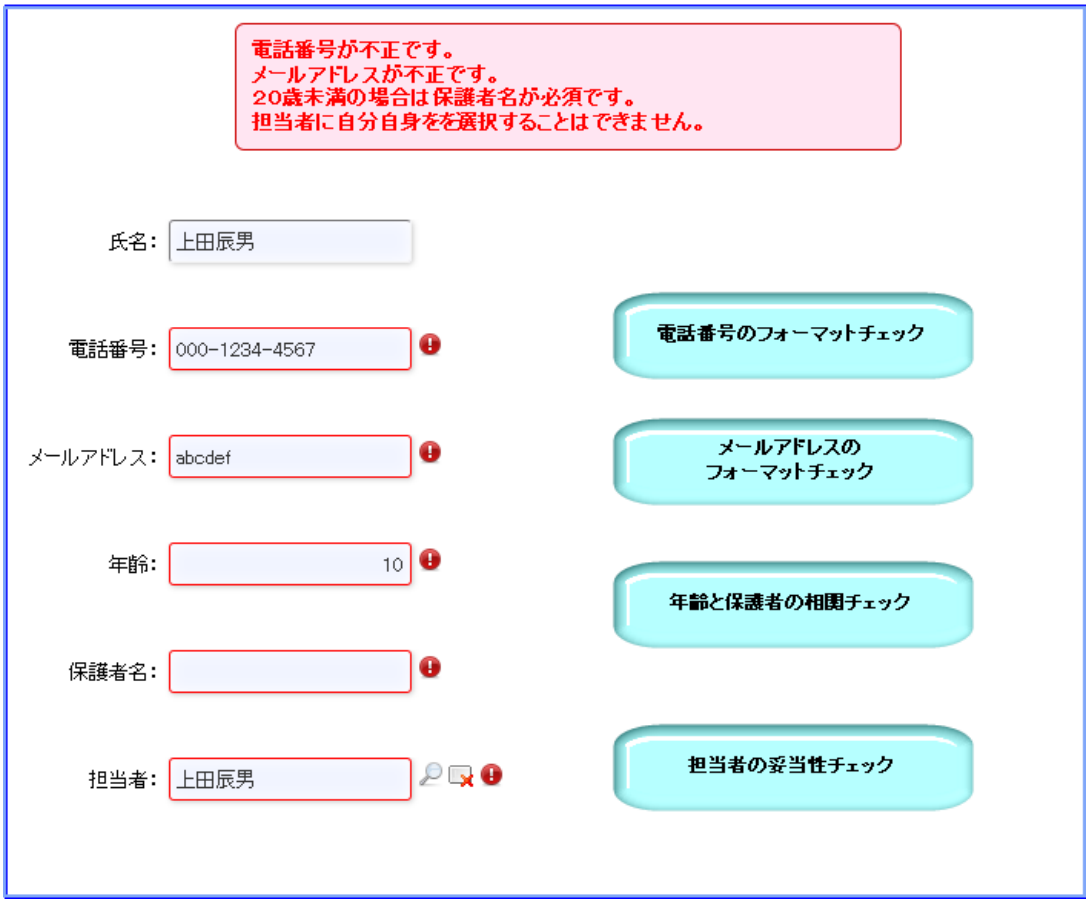
2.1 入力チェックプログラム

入力チェックプログラムを実装することで、アイテムに設定されている入力チェックプロパティの値をチェック実行時に動的に決定したり、各アイテムのフィールドの入力値に対し、アプリケーション独自のチェックを行い標準の入力チェックエラーの時と同じ形式で画面にエラー通知表示をすることが可能になります。

プログラムを作成しアプリ管理者用のメニューからユーザプログラムを登録すると、入力チェック処理時に該当プログラムが呼び出され実行されます。

入力チェックプログラムは一つのアプリケーション履歴に対して1プログラムが指定可能です。
入力チェックプログラムはスクリプト開発モデルで作成します。

- ユーザプログラムで4つの入力チェックを行っている例



2.2 後処理プログラム

後処理プログラムを実装することで、アプリケーションデータの登録・更新・削除処理後に動作する独自の処理をアプリケーションに組み込むことができます。

プログラムを作成しアプリ管理者用のメニューからユーザプログラムを登録すると、アプリケーションデータの登録・更新・削除処理が行われた後に同一トランザクション内で該当プログラムが呼び出され実行されます。

後処理プログラムは複数登録することができ、登録時に設定した順番で実行されます。
後処理プログラムは JavaEE 開発モデルまたはスクリプト開発モデルで作成します。

2.3 PDF出力処理プログラム

参照画面を PDF 出力する API を提供しています。PDF 出力機能は、Web アプリケーションサーバ内にインストールされた「wkhtmltopdf」を利用します。

wkhtmltopdf > http://code.google.com/p/wkhtmltopdf/

そのため、セットアップガイドに記載の通り、「wkhtmltopdf」をインストールしてください。
参照画面を PDF 出力する API では、以下のパラメータが必要です。

- アプリケーション ID
- ユーザデータ ID

参照画面を PDF 出力する API は、JavaEE 開発モデルまたはスクリプト開発モデルで利用できます。

3 入力チェックプログラム

3.1 入力チェックのパターン

入力チェックプログラムでは、以下の2つのパターンの実装が可能です。
チェックの内容に合わせて指定の関数に処理を実装します。

1. 入力チェックの内容をアプリケーション固有のルールに基づいて動的に変更する。
この場合は、フォーム作成時にフォームデザイナのプロパティ画面でアイテムに設定した入力チェックの値を変更するための処理を実装します。
入力チェック処理自体は各アイテムのものを利用するのでチェック処理を実装する必要はありません。
実装についての詳細は、後述の「3.2.3 入力チェックプロパティを設定する」をご参照ください。
例）
- あるアイテムの入力値によって、他のアイテムが必須であるかどうかを決定したい。
- ログインユーザの役職によってあるフィールドの上限値を動的に変更したい。

2. アプリケーション固有の入力チェックをする。
この場合は、入力チェック処理を独自に実装する必要があります。
例）
- 入力値に対してフィールド固有のフォーマットチェックをしたい。
- アイテム間で入力値の相関チェックをする
- 入力値に対してマスタの存在チェックをしたい。

実装についての詳細は、後述の「3.2.4 アプリケーション固有の入力チェックをする」をご参照ください。

また、入力チェックプログラムではエラーメッセージの表示順序を指定する機能も用意されています。
実装についての詳細は、後述の「3.2.5 エラーメッセージ表示順を指定する」をご参照ください。

3.2 入力チェックプログラムの実装

3.2.1 実装規約

プログラムを実装する場合、下記の制約に従って実装する必要があります。

- スクリプト開発モデルで実装すること

- 以下のメソッドのすべてを実装すること
  - getCustomProperties(formaParam, sendParam) メソッド
  - validate(formaParam, sendParam)メソッド
  - getErrorDisplayOrders(formaParam) メソッド

3.2.2 共通パラメータ

入力チェックのメソッドに渡される共通のパラメータ情報です。

3.2.2.1 formaParamオブジェクト

フォルマの基本情報を保持します。

プロパティの概要	
loginGroupId (String)	ログイングループ ID
loginUserCd (String)	ログインユーザ CD
applicationId (String)	アプリケーション ID
applicationNo (Number)	アプリケーションバージョン NO
insertId (String)	データ登録 ID
applicationType (String)	アプリケーション種別 アプリケーション種別は FRApplicationManager オブジェクトに定義されています。 APPLICATION_TYPE_STD：標準 APPLICATION_TYPE_IMW：IM-Workflow
appPageType (String)	アプリケーションページ種別 アプリケーションページ種別は FRApplicationPageInfo オブジェクトに定義されています。 REGISTRATION：登録・申請処理 EDIT：更新・再申請処理 POSTSCRIPT：承認処理 REFERENCE：参照 REFERENCE_EDIT：参照時の更新処理(IM-Workflow 時のみ)

3.2.2.2 sendParamオブジェクト

画面から渡されたリクエスト情報を保持する送信パラメータオブジェクトです。
明細テーブルアイテムを除く入力アイテムのパラメータ名はフィールド識別 ID になります。
明細テーブルアイテムのパラメータ名はテーブル識別 ID です。

アプリケーション種別が IM-Workflow の場合は、各種一覧画面から呼び出された際に受け取ることができる IM-Workflow リクエストパラメータ情報を含みます。取得可能な IM-Workflow リクエストパラメータの詳細は「IM-Workflow プログラミングガイド」をご参照ください。

アプリケーションの登録データの取得は、登録データ情報管理 API を利用しても取得が可能です。
取得方法については、後述の「6.1.2.1 データの取得」をご参照ください。
但し、送信パラメーまたは API の取得データには、ファイルアップロードアイテムのアップロードファイル情報は含まれません。

■ 入力アイテムのデータ型と取得値の対比

入力アイテムのデータ型により取得値は以下のように設定されています。

アイテムのデータ型	取得値	該当する主なアイテム
文字列	文字列	文字列、複数行文字列、(関数※1)(一覧選択※1)、チェックボックス※2、ラジオボタン、セレクトボックス、リストボックス※2
数値	数値の文字列	数値、(関数※1)、(一覧選択※1)
日付 または タイムスタンプ	1970 年 1 月 1 日 0 時 0 分 0 秒 0 ミリ秒からの通算ミリ秒の数値文字列	日付、期間、(関数※1)、(一覧選択※1)

※1 関数、一覧選択アイテムは取得値の設定によりアイテムのデータ型が決定します。

※2 複数項目選択可能なアイテム(チェックボックス、リストボックス)は選択された値をカンマ区切りにして値が設定されます。

◆ 明細テーブルアイテムの取得値

値は1レコード(行)毎に各列のフィールド識別IDと値をマッピングしたオブジェクトの配列です。

各列のデータ型は列タイプに依存します。

3.2.3 入力チェックプロパティを設定する

getCustomProperties(formaParam, sendParam)メソッド

このメソッドでは、入力チェックで利用されるアイテムの入力チェックプロパティの値をアプリケーション固有のルールに基づいて設定し、呼び出し元に返却します。

3.2.3.1 パラメータ

foramParam、sendParamについての詳細は「3.2.2　共通パラメータ」をご参照ください。

3.2.3.2 返却値

エラーフラグ error にはエラーの有無を設定し、data にはカスタムプロパティオブジェクトを設定して返却します。
カスタムプロパティの指定がない場合は data に空のオブジェクトを設定してください。

■　返却オブジェクト

<table>
<tr><th colspan="2">プロパティ詳細</th></tr>
<tr><td>error
（Boolean）</td><td>エラーフラグ
処理中エラーが発生した場合は true を設定、 以外は false を設定します。</td></tr>
<tr><td>errorMessage
(String)</td><td>エラーメッセージ</td></tr>
<tr><td>data
(Object)</td><td>カスタムプロパティオブジェクト
フィールド識別 ID をキーに入力チェックプロパティオブジェクトを設定します。
<table>
<tr><td>%フィールド識別 ID1%
(String)</td><td>入力プロパティオブジェクト
(Object)
<table>
<tr><td>%プロパティ 1%
（下記の入力プロパティ一覧参照）
%プロパティ 2%
・・・・・・・・・・・・
%プロパティn%</td></tr>
</table></td></tr>
<tr><td>%フィールド識別 ID2%</td><td>入力プロパティオブジェクト</td></tr>
<tr><td>%フィールド識別 IDn%</td><td>入力プロパティオブジェクト</td></tr>
</table></td></tr>
</table>

※ファイルアップロードアイテムはフィールド識別 IDを持たないため、ImFormaUtil オブジェクトの getFileUploadId メソッドにより取得される ID をセットします。
ImFormaUtilオブジェクトについての詳細は後述の「3.3 共通処理　ImFormaUtilオブジェクト」をご参照ください。

3.2.3.3 入力チェックプロパティー覧

FRItems オブジェクトでは入力チェックプロパティを定義しています。

プロパティ	チェック内容	設定値
FRItems.PROP_REQUIRED	必須	Boolean
FRItems.PROP_ONLY_ASCII	英数字のみ	Boolean
FRItems.PROP_MAX_LENGTH	最大入力値	Number
FRItems.PROP_MIN_LENGTH	最少入力値	Number
FRItems.PROP_PERMIT_MINUS	負数入力許可	Boolean
FRItems.PROP_PERMIT_DECIMAL	少数入力許可	Boolean
FRItems.PROP_DECIMAL_SIZE	少数部最大入力桁数	Number
FRItems.PROP_MIN_FILE_AMOUNT	最少入力ファイル数	Number
FRItems.PROP_MAX_FILE_AMOUNT	最大入力ファイル数	Number

3.2.3.4 実装例

```
function getCustomProperties(formaParam, sendParam) {
Debug.print("********** 入力チェックサンプルプログラム:カスタムプロパティを設定 **********");

    // フィールド(telnumber, age, startdate, userSelect)に対して入力チェックプロパティを設定する。
     var customPropObj = {telnumber: {}, age : {}, startdate: {}, userSelect : {}};
     if (xxxxxxxx) {  // 設定する条件句
       // "telnumber" 英数字のみを false
       customPropObj["telnumber"][FRItems.PROP_ONLY_ASCII] = false;
     }
     if (xxxxxxxx) {
       //"age" 最大入力値を 40 に設定
       customPropObj["age"][FRItems.PROP_MAX_LENGTH] = 40;
     }
     if (xxxxxxxx) {
       //"startdate" 必須を false
        customPropObj["startdate"][FRItems.PROP_REQUIRED] = false;
     }
     If (xxxxxxxx) {
       //" userSelect " 必須を false
       customPropObj["userSelect"][FRItems.PROP_REQUIRED] = false;
     }
     // ファイルアップロードアイテムの入力チェックプロパティを設定する
     If (xxxxxxxx) {
        var file_id = ImFormaUtil.getFileUploadId(0);
        customPropObj[file_id] = {};
        //ファイルアップロード 最少入力ファイル数を0
        customPropObj[file_id][FRItems.PROP_MIN_FILE_AMOUNT] = 0;
     }
     return { error : false , data : customPropObj};

     // カスタムプロパティの指定がない場合
     // return   { error : false , data : {} };

    //処理中にエラーが発生した場合
    // return   { error : true ,errorMessage : xxxxx };
}
```

3.2.4 アプリケーション固有の入力チェックをする

validate(formaParam, sendParam, resultObj)メソッド

このメソッドでは、アプリケーション独自の入力チェック処理を記述して、入力チェック処理結果を呼び出し元に返却します。

3.2.4.1 パラメータ

- foramParam、sendParamについての詳細は「3.2.2 共通パラメータ」をご参照ください。

- resultObj・・・返却オブジェクト

  入力チェックエラー情報を設定して返却するためのオブジェクトです。

プロパティ詳細	
error （Boolean）	エラーフラグ 入力チェックエラーがある場合は true を設定、 以外は false を設定します。
systemError （Boolean）	システムエラーフラグ システムエラー（入力チェック以外のエラー）がある場合は true を設定、 以外は false を設定します。
errorMessageList （Array）	エラーメッセージ情報配列 ◆ エラーメッセージ情報オブジェクト(Object) checkid：入力チェック ID errorMessage：エラーメッセージ
errorInfoList （Array）	エラー情報配列 ◆ エラー情報オブジェクト(Object) checkid：入力チェック ID itemType：アイテムタイプ acceptErrorObj：エラー通知オブジェクト

3.2.4.2 返却値

- 入力チェックエラーがなかった場合

  引数で渡された返却オブジェクト resultObj をそのまま返却します。

- 入力チェックエラーがあった場合

  ImFormaUtil オブジェクトを利用して返却オブジェクト resultObj にエラー情報を設定します。

  ユーザ画面へのエラー情報の通知方法は以下の二つの方法があります。

  - エラーメッセージのみを画面上部に表示する。
  - エラーがあったアイテムに対しエラー通知表示をする。
    （アイテムを赤枠で囲みエラー通知アイコンを表示など、エラー通知表示内容はアイテムに依存します）

  エラー通知の方法に合わせてImFormaUtilオブジェクトの該当メソッドを使ってエラー情報を設定してください。

  ImFormaUtilオブジェクトについての詳細は後述の「3.3 共通処理 ImFormaUtilオブジェクト」をご参照ください。

3.2.4.3 実装例

```
function validate(formaParam, sendParam, resultObj) {
Debug.print("********** 入力チェックサンプルプログラム：ユーザバリデーション実行 **********");

    // 入力チェック① フィールド識別 ID「sample_field_1」に対するチェック
    if (checkOne(sendParam["sample_field_1"])) {
        errorMessage = 'フィールド1の値が不正です。';
        // 返却オブジェクトにエラー情報を設定
        ImFormaUtil.addValidationError(resultObj, errorMessage, 'check1', ["sample_field_1"]);
    }

    // 入力チェック② フィールド識別 ID「sample_field_2」、「sample_field_3」の相関チェック
    if (checkTwo(sendParam["sample_field_2"]), sendParam["sample_field_3"])) {
        errorMessage = 'フィールド2の値が〇〇〇の場合はフィールド3は XXX でなくてはいけません。';
        ImFormaUtil.addValidationError(resultObj, errorMessage, 'check2', ["sample_field_2",  "sample_field_3"] );
    }

    // 明細テーブル（テーブル識別 ID  comp1_table_1）内のフィールドに対するチェック
    var tableId = "comp1_table_1";
    // 明細テーブルデータの取得
    var tableData = sendParam[tableId];
    // 入力行数分チェック
    for (var i = 0, length = tableData.length; i < length; i++) {
       // 列数分チェック
        for (var input_id in tableData[i]) {
            var checkValue = tableData[i][input_id];
            if (checkValue == "") continue;
            //入力チェック③ 列1のフィールドに対するチェック
            if (input_id == "table_field_1") {
                if (checkRow1 (checkValue)) {
                    errorMessage = 'XXXXXXX ';
                    ImFormaUtil.addValidationError( resultObj, errorMessage, 'check3', [input_id], i );
                }
             //入力チェック④ 列 2 のフィールドに対するチェック
            } else if ((input_id == "table_field_2") {
                if (checkRow2 (checkValue)) {
                    errorMessage = 'XXXXXXX ';
                    ImFormaUtil.addValidationError( resultObj, errorMessage, 'check4', [input_id], i );
                }
            } else if ( ・・・・・・・・・・・・・・・・・・・・・){
                ・・・・・・・・・・・・・・・・
            }
    }
    // 返却オブジェクトを返却
    return   resultObj;
}

function checkOne (inputValue){
.  // 「sample_field_1」に対する入力チェック処理を実装する。
}
function checkTwo (inputValue1, inputValue2) {
.    // 「sample_field_2」「sample_field_3」に対する入力チェック処理を実装する。
}
}
function checkRow1 (inputValue) {
  //明細テーブル内フィールド「table_field_1」に対する入力チェック処理を実装する。
}
function checkRow2 (inputValue) {
  //明細テーブル内フィールド「table_field_2」に対する入力チェック処理を実装する。
}
```

上記の実装例では、4つの入力チェックを行っています。

【主な処理内容】
1. 送信パラメータ sendParam から入力値を受け取って順に妥当性を検証
2. 各入力チェックでは、validate メソッド内で一意になる固定の入力チェック ID（上記の例では check1~ check4）を付与
3. 入力チェックエラー時は返却オブジェクトに付与した入力チェック ID を含むエラー情報を設定

3.2.4.4 システムエラー時の動作仕様

処理中に入力チェック以外のエラーが発生した場合は、返却オブジェクトのシステムエラーフラグ systemError を true に設定してください。
システムエラーフラグが true の場合または処理中に予期しないエラーが発生した場合は、画面にはシステムエラーが発生した旨のメッセージのみが表示され入力チェックに関するエラーの通知は行われません。
その際 forma ログにはエラーに関する情報が出力されます。

3.2.5 エラーメッセージ表示順を指定する

getErrorDisplayOrders(formaParam)メソッド

このメソッドでは、入力チェックエラーがあった場合にエラーメッセージを表示する順番をアイテム単位で指定して呼び出し元に返却します。

表示順を指定しない場合のエラーメッセージの表示順は以下の通りです。
1. アイテムの重なり順（フォームデザイナのラベル一覧で表示される順番）
2. ユーザバリデーションの validate メソッドでエラー情報を登録した順

尚、このメソッドにより複数入力フィールドを持つアイテムの入力フィールドに対するアイテム内でのエラーメッセージの順序は、アイテムの入力チェックエラー処理に依存するため変更することはできません。
例）
期間アイテムが[開始日フィールド]→[終了日フィールド]の順にエラーを出力している場合、[終了日フィールド]→[開始日フィールド]の順に変更することができません。

3.2.5.1 パラメータ

foramParamについての詳細は「3.2.2 共通パラメータ」をご参照ください。

3.2.5.2 返却値

入力チェックエラーがあった場合にエラーメッセージを表示する順番で下記の ID をセットします。

- 標準で提供する入力チェックによるメッセージ
  アイテム ID をセットします。 アイテム ID は ImFormaUtil オブジェクトの getItemIdByInputId メソッドにフィールド識別 ID を渡すことで取得可能です。
  複数の入力フィールドを持つアイテムの場合は、任意の一つのフィールド識別 ID を設定してください。
  ファイルアップロードアイテムはフィールド識別 ID を持ちませんので、フィールド識別 ID の代わりに ImFormaUtil オブジェクトの getFileUploadId 関数により取得される ID をセットします。

- ユーザプログラムの validate メソッドで行われる入力チェックによるメッセージ
  入力チェックプログラムの validate メソッド内で付与した入力チェック ID をセットします。

表示順の指定がない場合は空の配列を返却してください。

3.2.5.3実装例

```
function getErrorDisplayOrder(formaParam) {
Debug.print("********** 入力チェックサンプルプログラム：エラーメッセージ表示順を指定 **********");
    var arry = [];

    //標準で提供する入力チェックによるメッセージ
    //フィールド識別 ID「sample_field_1」
    arry.push(ImFormaUtil.getItemIdByInputId("sample_field_1"));
    arry.push(ImFormaUtil.getItemIdByInputId("sample_field_3"));

    //標準で提供する入力チェックによるメッセージ(ファイルアップロードアイテムの場合)
    arry.push(ImFormaUtil.getItemIdById(ImFormaUtil.getFileUploadId(0)));

    // validate メソッドで行われる入力チェックによるメッセージ
    //入力チェック ID「check1」
    arry.push("check1");

    arry.push(xxxxxxxxxxxxxxxxxx);
    arry.push(xxxxxxxxxxxxxxxxxx);
    arry.push(xxxxxxxxxxxxxxxxxx);
    ・・・・・・・・・・・・・・・・・・・・

    return arry;
}
```

3.3 共通処理 ImFormaUtilオブジェクト

ImFormaUtil オブジェクトは、入力チェックプログラム内で使用するためのメソッドを提供する JavaScriptAPI です。
このオブジェクトが提供するメソッドは入力チェックプログラム内でしか使用することができません。

3.3.1 入力チェックエラー情報設定メソッド

void addValidationError(resultObj, errorMessage, checkid, inputIds, index)

引数で受け取った情報を元に、返却オブジェクトにエラー情報を設定します。
エラーがあったアイテムに対しエラーメッセージの表示とエラー通知表示（アイテムを赤枠で囲みエラー通知アイコンを表示など、エラー通知表示内容はアイテムに依存）を行う場合にこのメソッドを利用します。
但し、エラー通知表示の実装(input.html の AcceptError/ClearError メソッド)はアイテム毎にそれぞれで実装されているため、この関数を利用してエラー通知表示を行う対象になるのは基本的に標準で提供しているアイテムです。
独自に作成したアイテムに対し、エラー通知表示を行いたい場合は、後述の「3.4 追加したアイテムにエラー通知表示を行う方法」をご参照ください。

■ パラメータ

resultObj (Object)	返却オブジェクト validate メソッドの引数で受け取った返却オブジェクトを設定します。
errorMessage (String)	エラーメッセージ
checkid (String)	入力チェック ID
inputIds (Array)	エラー対象のフィールド識別 ID の配列
index (Number)	エラー対象のアイテムが明細テーブルの場合、エラーになった行のインデックスを設定します。 明細テーブル以外のアイテムの場合はこのパラメータは必要ありません。

■ 返却値

返却値はありません。

3.3.2 入力チェックエラーメッセージ設定メソッド

void addValidationErrMsg(resultObj, errorMessage, checkid)

引数で受け取った情報を元に、返却オブジェクトにエラー情報を設定します。
エラーがあったアイテムに対しエラー通知表示を行わずにエラーメッセージのみを画面上部に表示する場合にこの関数を利用します。

■ パラメータ

resultObj (Object)	返却オブジェクト validate メソッドの引数で受け取った返却オブジェクトを設定します。
errorMessage (String)	エラーメッセージ
checkid (String)	入力チェック ID

■ 返却値
返却値はありません。

3.3.3 アイテムID取得メソッド

String getItemIdById (inputId)

引数で受け取ったフィールド識別 ID を持つアイテム ID またはファイルアップロード ID（下記を参照）に該当するアイテム ID を返却します。

■ パラメータ
inputid にはフィールド識別 ID またはファイルアップロード ID を指定します。

■ 返却値
アイテム ID

3.3.4 ファイルアップロードID取得メソッド

String getFileUploadId (index)

ファイルアップロードアイテムはフィールド識別 ID を持たないため、入力チェック処理内でアイテムを識別するための ID を生成して返却します。

■ パラメータ
index にはフォーム内に出現するファイルアップロードアイテムの順番を指定します。
一番目のファイルアップロードアイテムの index には「0」を指定します。
※ここでいう出現順とは、アイテムの重なり順（フォームデザイナでラベル一覧を表示した場合に表示される順番）を意味します。

■ 返却値
ファイルアップロード ID

3.4 追加したアイテムにエラー通知表示を行う方法

追加したアイテムが input.html の AcceptError/ClearError メソッドで独自のエラー通知処理を行っている場合には、addValidationError メソッドを使用して返却オブジェクトにエラー情報を設定することができません。
※独自に追加したアイテムでも文字列アイテムなど、標準で提供しているアイテムと同じエラー通知処理を行っている場合は addValidationError メソッドを使用することができます。

独自のエラー通知処理を行っているアイテムに対し、入力エラーの際にエラー通知表示を行いたい場合は、addValidationError メソッドの代わりに返却オブジェクトにエラー情報を設定する処理を実装する必要があります。

3.4.1 エラー情報の設定

返却オブジェクトにはエラーメッセージとエラー情報を追加します。
エラーメッセージの追加は addValidationErrMsg メソッドが利用できます。

■ 返却オブジェクトの構成

<table>
<tr><th colspan="2">プロパティ詳細</th></tr>
<tr><td>error
（Boolean）</td><td>エラーフラグ
入力チェックエラーがある場合は true を設定、 以外は false を設定します。</td></tr>
<tr><td>systemError
（Boolean）</td><td>システムエラーフラグ
システムエラー（入力チェック以外のエラー）がある場合は true を設定、 以外は false を設定します。</td></tr>
<tr><td>errorMessageList
（Array）</td><td>エラーメッセージ情報配列
◆ エラーメッセージ情報オブジェクト(Object)
<table>
<tr><td>checkid</td><td>入力チェック ID</td></tr>
<tr><td>errorMessage</td><td>エラーメッセージ</td></tr>
</table></td></tr>
<tr><td>errorInfoList
（Array）</td><td>エラー情報配列
◆ エラー情報オブジェクト(Object)
<table>
<tr><td>checkid</td><td>入力チェック ID</td></tr>
<tr><td>itemType</td><td>アイテムタイプ</td></tr>
<tr><td>acceptErrorObj</td><td>エラー通知オブジェクト</td></tr>
</table></td></tr>
</table>

■ 実装しなくてはいけない処理

1. エラーフラグ error が false の場合は、true に変更
2. エラーメッセージ情報 errorMessageList にエラーメッセージオブジェクトを追加
3. エラー情報配列 errorInfoList にエラー情報を追加
   acceptErrorObj には該当アイテムの type.js の validate メソッドでエラーの場合に返却しているオブジェクトと同じ構造のオブジェクトを設定する必要があります。

4 後処理プログラム

IM-FormaDesigner では、アプリケーションデータの登録、更新、削除などデータの操作が行われた後に任意のプログラムを実行することができる機能を提供しています。
アプリケーションの種別を問わずデータの操作が行われたタイミングで後処理は実行されます。
IM-Workflow の処理に合わせて後処理を実行したい場合には、IM-Workflow のユーザプログラム機能をご利用ください。

4.1 後処理プログラムの実装

4.1.1 JavaEE開発モデル

JavaEE 開発モデルにおいて、後処理プログラムを実装する手順を示します。

4.1.1.1 実装規約

JavaEE 開発モデルにおいて、後処理プログラムを実装する場合、下記の制約に従って実装する必要があります。

- jp.co.intra_mart.foundation.forma.userprogram.PostProcessExecutor を継承すること
- プログラムを実行するタイミングに合わせて以下のメソッドを実装すること
  - データの登録時
    regist(PostProcessParameter formaParam, Map<String, Object> sendParam)メソッド
  - データの更新時
    update(PostProcessParameter formaParam, Map<String, Object> sendParam) メソッド
  - データの削除時
    remove(PostProcessParameter formaParam) メソッド

4.1.1.2 パラメータ

各メソッドに渡されるパラメータが保持する情報は以下の通りです。

4.1.1.2.1 **jp.co.intra_mart.foundation.forma.userprogram.PostProcessParameter**

フォルマの基本情報を保持するクラスです。

メソッドの概要	
String getLoginGroupId()	ログイングループ ID を返却します。
String getLoginUserCd()	ログインユーザ CD を返却します。
String getApplicationId()	アプリケーション ID を返却します。
long getApplicationNo()	アプリケーションバージョン NO を返却します。
String getInsertId()	データ登録 ID を返却します。

ApplicationType getApplicationType()	アプリケーション種別を返却します。 アプリケーション種別は以下の列挙型で定義されています。 jp.co.intra_mart.foundation.forma.type.ApplicationType STD：標準 IMW：IM-Workflow
ApplicationPageType getAppPageType()	アプリケーションページ種別を返却します。 アプリケーションページ種別は以下の列挙型で定義されています。 jp.co.intra_mart.foundation.forma.type.ApplicationType REGISTRATION：登録・申請処理 EDIT：更新・再申請処理 POSTSCRIPT：承認処理 REFERENCE：参照 REFERENCE_EDIT：参照時の更新処理(IM-Workflow 時のみ)

4.1.1.2.2 sendParamマップ

画面から渡されたリクエスト情報を保持する送信パラメータ Map です。
明細テーブルアイテムを除く入力アイテムのキーはフィールド識別 ID になります。
明細テーブルアイテムのキーはテーブル識別 ID です。

アプリケーション種別が IM-Workflow の場合は、各種一覧画面から呼び出された際に受け取ることができる IM-Workflow リクエストパラメータ情報を含みます。取得可能な IM-Workflow リクエストパラメータの詳細は「IM-Workflow プログラミングガイド」をご参照ください。

アプリケーションの登録データの取得は、登録データ情報管理 API を利用しても取得が可能です。
取得方法については、後述の「6.1.2.1 データの取得」をご参照ください。
但し、送信パラメーまたは API の取得データには、ファイルアップロードアイテムのアップロードファイル情報は含まれません。

■ 入力アイテムのデータ型と取得値の対比

入力アイテムのデータ型と取得値は以下の通りです。

アイテムのデータ型	取得値	該当する主なアイテム
文字列	文字列	文字列、複数行文字列、(関数※1)(一覧選択※1)、チェックボックス※2、ラジオボタン、セレクトボックス、リストボックス※2
数値	数値の文字列	数値、(関数※1)、(一覧選択※1)
日付 または タイムスタンプ	1970 年 1 月 1 日 0 時 0 分 0 秒 0 ミリ秒からの通算ミリ秒の数値文字列	日付、期間、(関数※1)、(一覧選択※1)

※1 関数、一覧選択アイテムは取得値の設定によりアイテムのデータ型が決定します。
※2 複数項目選択可能なアイテム(チェックボックス、リストボックス)は選択された値をカンマ区切りにして値が設定されます。

◆ 明細テーブルアイテムの取得値
取得値は1レコード(行)毎に各列のフィールド識別IDと値をマッピングした Map の List です。
各列のデータ型は列タイプに依存します。

4.1.1.3 返却値とエラー処理

返却値はありません。
処理中にエラーが発生した場合は、Exception を throw してください。
エラーが発生した場合は、処理を中断しエラー発生以前に行われたすべての処理がロールバックされます。

4.1.1.4 トランザクションの制御

後処理プログラムはトランザクション内で実行されるため、このプログラム中では DB トランザクション制御を行うことはできません。
実行プログラム中においてトランザクションのコミット、ロールバック等は行わないで下さい。

4.1.1.5 実装例

```
public class SamplePostProcess  extends PostProcessExecutor{

    /**
     * 登録処理を行った場合に実行されます。<BR>
     *
     * @param formaParam フォルマパラメータ
     * @param sendParam 送信パラメータ
     * @throws Exception 例外が発生
     */
    public void regist(PostProcessParameter formaParam, Map<String, Object> sendParam) throws Exception {
        System.out.println("********** JAVAEE 開発:登録後処理サンプルプログラム **********");
        // ここに登録処理後に実行する処理を記述します。
    }

    /**
     * 更新処理を行った場合に実行されます。<BR>
     *
     *
     * @param formaParam フォルマパラメータ
     * @param sendParam 送信パラメータ
     * @throws Exception 例外が発生
     */
    public void update(PostProcessParameter formaParam, Map<String, Object> sendParam) throws Exception {
        System.out.println("********** JAVAEE 開発:更新後処理サンプルプログラム**********");
        // ここに更新処理後に実行する処理を記述します。
      }

    /**
     * 削除処理を行った場合に実行されます。<BR>
     *
     * @param formaParam フォルマパラメータ
     * @throws Exception 例外が発生
     */
    public void remove(PostProcessParameter formaParam) throws Exception {
        System.out.println("********** JAVAEE 開発:削除後処理サンプルプログラム**********");
        // ここに削除処理後に実行する処理を記述します。
    }
```

4.1.2 スクリプト開発モデル

スクリプト開発モデルにおいて、後処理プログラムを実装する手順を示します。

4.1.2.1 実装規約

スクリプト開発モデルにおいて、後処理プログラムを実装する場合、下記の制約に従って実装する必要があります。

■ プログラムを実行するタイミングに合わせて以下のメソッドを実装すること

◆ データの登録時
regist(formaParam, sendParam)メソッド

◆ データの更新時
update(formaParam, sendParam) メソッド

◆ データの削除時
remove(formaParam) メソッド

4.1.2.2 パラメータ

4.1.2.2.1 formaParamオブジェクト

フォルマの基本情報を保持します。

プロパティの概要		
loginGroupId (String)	ログイングループ ID	
loginUserCd (String)	ログインユーザ CD	
applicationId (String)	アプリケーション ID	
applicationNo (Number)	アプリケーションバージョン NO	
insertId (String)	データ登録 ID	
applicationType (String)	アプリケーション種別 アプリケーション種別は FRApplicationManager オブジェクトに定義されています。	
	APPLICATION_TYPE_STD	標準
	APPLICATION_TYPE_IMW	IM-Workflow
appPageType (String)	アプリケーションページ種別 アプリケーションページ種別は FRApplicationPageInfo オブジェクトに定義されています。	
	REGISTRATION	登録・申請処理
	EDIT	更新・再申請処理
	POSTSCRIPT	承認処理
	REFERENCE	参照
	REFERENCE_EDIT	参照時の更新処理(IM-Workflow 時のみ)

4.1.2.2.2 sendParamオブジェクト

画面から渡されたリクエスト情報を保持する送信パラメータオブジェクトです。
明細テーブルアイテムを除く入力アイテムのパラメータ名はフィールド識別 ID です。
明細テーブルアイテムのパラメータ名はテーブル識別 ID です。

アプリケーション種別が IM-Workflow の場合は、各種一覧画面から呼び出された際に受け取ることができる IM-Workflow リクエストパラメータ情報を含みます。取得可能な IM-Workflow リクエストパラメータの詳細は「IM-Workflow プログラミングガイド」をご参照ください。

アプリケーションの登録データの取得は、登録データ情報管理 API を利用しても取得が可能です。
取得方法については、後述の「6.1.2.1 データの取得」をご参照ください。
但し、送信パラメーまたは API の取得データには、ファイルアップロードアイテムのアップロードファイル情報は含まれません。

■ 入力アイテムのデータ型と取得値の対比

入力アイテムのデータ型により取得値は以下のように設定されています。

アイテムのデータ型	取得値	該当する主なアイテム
文字列	文字列	文字列、複数行文字列、(関数※1)(一覧選択※1)、チェックボックス※2、ラジオボタン、セレクトボックス、リストボックス※2
数値	数値の文字列	数値、(関数※1)、(一覧選択※1)
日付 または タイムスタンプ	1970 年 1 月 1 日 0 時 0 分 0 秒 0 ミリ秒からの通算ミリ秒の数値文字列	日付、期間、(関数※1)、(一覧選択※1)

※1 関数、一覧選択アイテムは取得値の設定によりアイテムのデータ型が決定します。
※2 複数項目選択可能なアイテム(チェックボックス、リストボックス)は選択された値をカンマ区切りにして値が設定されます。

◆ 明細テーブルアイテムの取得値

値は1レコード(行)毎に各列のフィールド識別IDと値をマッピングしたオブジェクトの配列です。
各列のデータ型は列タイプに依存します。

4.1.2.3 返却値とエラー処理

処理結果オブジェクト
- error : エラーフラグ 処理に失敗した場合は true 、成功した場合は false
- errorMessage : エラーメッセージ 設定したメッセージはエラーログに出力されます。

エラー発生時には処理結果オブジェクトのエラーフラグを true にしてエラーメッセージを設定してください。
エラーフラグが true の場合は、処理を中断しエラー発生以前に行われたすべての処理がロールバックされます。

4.1.2.4 トランザクションの制御

後処理プログラムはトランザクション内で実行されるため、このプログラム中では DB トランザクション制御を行うことはできません。
実行プログラム中においてトランザクションのコミット、ロールバック等は行わないで下さい。

4.1.2.5 実装例

```
// 登録
function regist(formaParam, sendParam) {
      Debug.print("********** スクリプト開発:登録後処理サンプルプログラム **********");
      // ここに登録処理後に実行する処理を記述します。
      return {"error" : false};
      // エラーが発生した場合
      // return {"error" : true, "errorMessage" :"エラーメッセージ"};
}

// 更新
function update(formaParam, sendParam) {
   Debug.print("**********  スクリプト開発:更新後処理サンプルプログラム **********");
   // ここに更新処理後に実行する処理を記述します。
      return {"error" : false};
}

// 削除
function remove(formaParam) {
      Debug.print("**********  スクリプト開発:削除後処理サンプルプログラム **********");
   // ここに削除処理後に実行する処理を記述します。
      return {"error" : false};
}
```

5 PDF出力処理プログラム

IM-FormaDesigner では、参照画面を PDF 出力する API を提供しています。PDF 出力機能は、Application Runtime と同じ環境内にインストールされた「wkhtmltopdf」を利用します。参照画面を PDF 出力する API では、以下のパラメータが必要です。

- アプリケーション ID
- ユーザデータ ID

以下では、PDF 出力処理の実装例を紹介します。また IM-Workflow Ver.7.2 の案件終了処理等、非同期設定で動作する到達処理内では、以下に紹介している通り、実装方法が異なるため、注意してください。

5.1 PDF出力処理の仕組みと応用

はじめに、PDF 出力処理は、API を実行するログインユーザで実行されます。そのため、ログインユーザに参照権限がなければ PDF 出力できません。またログインユーザのロケール設定やカラーパターン設定が影響します。

PDF 出力機能は、Application Runtime と同じ環境内にインストールされた「wkhtmltopdf」を利用します。「wkhtmltopdf」には、HTML のレンダリングエンジンに Safari や Chrome で利用されている Webkit が組み込まれています。つまり、「wkhtmltopdf」は Application Runtime と同じ環境内で参照画面の HTML を描画して PDF に出力しています。そのため、Web ブラウザで表示される見た目と PDF ファイルの見た目が異なる場合があります。たとえば、PDF に埋め込まれるフォントは、「wkhtmltopdf」がインストールされた環境に依存します。

API では「wkhtmltopdf」の各種オプションを設定できます。API にオプションを指定しない場合、以下のオプションが付与しています。

- disable-external-links : リンクを無効にします。
- print-media-type : 印刷モードで出力します。
  ※標準では、forma/css/print.css が有効になり、ボタンを非表示指定します。

その他利用可能なオプションについては http://code.google.com/p/wkhtmltopdf/ のマニュアルまたはhelpコマンド (下記参照) をご覧ください。

```
$ wkhtmltopdf -H
```

PDF 出力が正常に実行できた場合、PDF ファイルは work/tmp/forma/print/ディレクトリに一時ファイルとして出力されます。一時ファイルとして出力された PDF ファイルは、使用用途に応じて Storage に永続化したり、クライアントにダウンロードされることを期待しています。
そのため、work/forma/tmp/print/ ディレクトリに出力された PDF ファイルは、API 使用後に明示的に削除するようにしてください。
※もし削除せず残っていた場合、Web アプリケーションサーバが正常に停止した場合、停止時に上記ファイルは自動的に削除されます。
※Web アプリケーションサーバが異常停止した場合、削除されません。work/forma/tmp/print/ にゴミが残っている場合は、適宜削除してください。

5.2 JavaEE開発モデルでの実装例

JavaEE 開発モデルにおいて、PDF 出力処理を実装する手順を示します。

5.2.1 実装規約

JavaEE 開発モデルにおいて、PDF 出力処理を実装する場合、下記の制約に従って実装する必要があります。

- jp.co.intra_mart.foundation.forma.ApplicationPDFConverter (以下 ApplicationPDFConverter と略す)を利用します
- 出力された PDF ファイルはできる限り出力処理後に削除すること推奨します

5.2.2 実装例 1: wkhtmltopdfオプション指定なし

```
import java.io.File;

import jp.co.intra_mart.foundation.forma.ApplicationPDFConverter;
import jp.co.intra_mart.foundation.forma.exception.FormaSystemException;

(省略)

    // パラメータ
    final String applicationId = "XX"; // アプリケーション ID
    final String insertId      = "XX"; // ユーザデータ ID

    // PDF 出力処理
    final ApplicationPDFConverter pdfConverter = new ApplicationPDFConverter();
    File pdfFile = null;
    try {
        pdfFile = pdfConverter.createPDF(applicationId, insertId);

        : (ファイル操作処理)

    } catch(final FormaSystemException e) {
        // TODO エラー処理
        e.printStacktrace();
    } finally {
        // 出力された PDF ファイルを削除
        if (pdfFile != null && pdfFile.exists()) {
            pdfFile.delete();
        }
    }

(省略)
```

5.2.3 実装例 2: wkhtmltopdfオプション指定あり

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import jp.co.intra_mart.foundation.forma.ApplicationPDFConverter;
import jp.co.intra_mart.foundation.forma.exception.FormaSystemException;

(省略)

	// パラメータ
	final String applicationId = "XX"; // アプリケーション ID
	final String insertId      = "XX"; // ユーザデータ ID

	// wkhtmltopdf オプション
	final List<String> options = new ArrayList<String>();
	// リンク無効
	options.add("--disable-external-links");
	// 印刷モード
	options.add("--print-media-type");

	// PDF 出力処理
	final ApplicationPDFConverter pdfConverter = new ApplicationPDFConverter();
	File pdfFile = null;
	try {
		pdfFile = pdfConverter.createPDF(applicationId, insertId, options);

		: (ファイル操作処理)

	} catch(final FormaSystemException e) {
		// TODO エラー処理
		e.printStacktrace();
	} finally {
		// 出力された PDF ファイルを削除
		if (pdfFile != null && pdfFile.exists()) {
			pdfFile.delete();
		}
	}

(省略)
```

5.3 スクリプト開発モデルでの実装例

スクリプト開発モデルにおいて、PDF 出力処理を実装する手順を示します。

5.3.1 実装規約

スクリプト開発モデルにおいて、PDF 出力処理を実装する場合、下記の制約に従って実装する必要があります。

- FRApplicationPDFConverter を利用します
- 出力された PDF ファイルはできる限り出力処理後に削除すること推奨します

5.3.2 実装例 1: wkhtmltopdfオプション指定なし

```
（省略）
    // パラメータ
    var applicationId = "XX"; // アプリケーション ID
    var insertId      = "XX"; // ユーザデータ ID

    // PDF 出力処理
    var pdfConverter = new FRApplicationPDFConverter();
    var result = pdfConverter.createPDF(applicationId, insertId);
    if (result.error) {
        // TODO エラー処理
        Debug.console(result);
    } else {
        // PDF ファイル
        var pdfFile = result.data;

        : (ファイル操作処理)

        // 出力された PDF ファイルを削除
        pdfFile.remove();
    }
（省略）
```

5.3.3 実装例 2: wkhtmltopdfオプション指定あり

```
(省略)
	// パラメータ
	var applicationId = "XX"; // アプリケーション ID
	var insertId      = "XX"; // ユーザデータ ID

	// wkhtmltopdf オプション
	var options = [];
	// リンク無効
	options.push("--disable-external-links");
	// 印刷モード
	options.push("--print-media-type");

	// PDF 出力処理
	var pdfConverter = new FRApplicationPDFConverter();
	var result = pdfConverter.createPDF(applicationId, insertId);
	if (result.error) {
		// TODO エラー処理
		Debug.console(result);
	} else {
		// PDF ファイル
		var pdfFile = result.data;

		: (ファイル操作処理)

		// 出力された PDF ファイルを削除
		pdfFile.remove();
	}
(省略)
```

5.4 IM-Workflow Ver.7.2 アクション処理、到達処理内での実装方法

IM-Workflow Ver.7.2 において、案件終了処理、到達処理、メール送信処理の同期/非同期制御の設定が「非同期」のとき、コンストラクタに PDF 出力処理を行うユーザのログインユーザ CD とログイングループ ID を指定してください。

■ JavaEE 開発モデルでの実装例

```
// PDF 出力処理
final ApplicationPDFConverter pdfConverter = new ApplicationPDFConverter(loginUserCd, loginGroupId);
```

■ スクリプト開発モデルでの実装例

```
// PDF 出力処理
var pdfConverter = new FRApplicationPDFConverter(loginUserCd, loginGroupId);
```

[注意] 上記に示すコンストラクタは、IM-FormaDesigner Ver.7.2 専用の API です。IM-FormaDesigner for Accel Platform では上記コンストラクタは廃止され、引数なしのコンストラクタを利用します。

次に、PDF 出力に必要なパラメータとして、アプリケーション ID とユーザデータ ID が必要です。案件終了処理内では、IM-Workflow のパラメータから PDF 出力に必要なパラメータのアプリケーション ID が取得できません。そのため、下記に示すように、IM-Workflow 連携情報管理クラスの API を利用してアプリケーション ID を取得してください。

■ JavaEE 開発モデルでの実装例

```
import jp.co.intra_mart.foundation.forma.ImwRelationManager;
import jp.co.intra_mart.foundation.forma.model.imw.ImwMatterInfoModel;
import jp.co.intra_mart.foundation.forma.exception.FormaSystemException;
import jp.co.intra_mart.foundation.forma.exception.FormaValidationException;

（省略）

    // パラメータ
    String applicationId = null; // アプリケーション ID
    final String insertId = userDataId; // ユーザデータ ID は、IM-Workflow のユーザデータ ID に一致

    ImwRelationManager imwRelationManager = new ImwRelationManager(loginUserCd, loginGroupId);
    ImwMatterInfoModel imwMatter;
    try {
        imwMatter = imwRelationManager.getImwMatter(insertId);
        applicationId = imwMatter.getApplicationId();
    } catch (FormaValidationException e) {
        // TODO エラー処理
    } catch (FormaSystemException e) {
        // TODO エラー処理
    }

（省略）
```

■　スクリプト開発モデルでの実装例

```
(省略)

    // パラメータ
    var applicationId = ""; // アプリケーション ID
    var insertId        = userDataId; // ユーザデータ ID は、IM-Workflow のユーザデータ ID に一致

    var imwRelationManager = new FRImwRelationManager(loginUserCd, loginGroupId);
    var imwMatter = imwRelationManager.getImwMatter(insertId);
    if (imwMatter.error) {
        // TODO エラー処理
    } else {
        applicationId = imwMatter.data.application_id;
    }

(省略)
```

6 登録データ情報管理API

この章では、アプリケーションの登録データを取得または操作する方法・手順について解説します。
この API で提供するメソッドはアプリケーションのデータをデータベースから取得、操作するものであり、権限によるチェックは行いませんのでご注意ください。

6.1 JavaEE開発モデル

アプリケーションデータを登録または検索・削除するための下記のクラスが用意されています。

jp.co.intra_mart.foundation.forma.ApplicationDataManager

6.1.1 コンストラクタ

■ コンストラクタ①

・ **ApplicationDataManager(String loginUserCd, String loginGroupId)**

このコンストラクタを利用してマネージャオブジェクトを生成した場合、データ操作を行った際に後処理の設定があった場合は後処理が実行されます。

◆ パラメータ

loginUserCd ログインユーザコード
loginGroupId ログイングループID

■ コンストラクタ②

・ **ApplicationDataManager(String loginUserCd, String loginGroupId, boolean execPostProcFlg)**

このコンストラクタを利用してマネージャオブジェクトを生成時にデータ操作を行った際の後処理の実行有無を指定することができます。

◆ パラメータ

loginUserCd ログインユーザコード
loginGroupId ログイングループID
execPostProcFlg 後処理実行フラグ 後処理を実行する場合は true、しない場合は false

6.1.2 主なメソッド

6.1.2.1 データの取得メソッド

Map<String, Object> getItemData (String applicationId, String insertId, boolean systemData)

指定されたデータ登録 ID の登録情報を返却します。

◆ パラメータ

applicationId アプリケーション ID
insertId ログデータ登録 ID
systemData システム情報の取得の有無 取得する場合は true、しない場合は false

◆ 返却値
登録情報マップ※詳細は後述の「6.1.2.5 登録情報マップ」をご参照ください。

6.1.2.2 データの登録メソッド

void insertItemData(String applicationId, long applicationNo, String insertId, Map<String, Object> itemData)

アプリケーションの登録情報をテーブルに登録します。

◆ パラメータ
applicationId アプリケーションID
applicationNo アプリケーションバージョンNO
insertId 登録データID
itemData 登録情報マップ※詳細は後述の「6.1.2.5 登録情報マップ」をご参照ください。
◆ 返却値
なし

後処理内でこのメソッドを使用する場合、さらに後処理を実行する必要がない場合はコンストラクタ②の execPostProcFlg を false にしてマネージャオブジェクトを生成してください。

6.1.2.3 データの更新メソッド

void udpateItemData(String applicationId, long applicationNo, String insertId, Map<String, Object> itemData)

指定された登録データ ID の登録情報を更新します。

◆ パラメータ
applicationId アプリケーションID
applicationNo アプリケーションバージョンNO
insertId 登録データID
itemData ※詳細は後述の「6.1.2.5 登録情報マップ」をご参照ください。
◆ 返却値
なし

後処理内でこのメソッドを使用する場合、さらに後処理を実行する必要がない場合はコンストラクタ②の execPostProcFlg を false にしてマネージャオブジェクトを生成してください。

6.1.2.4 データの削除メソッド

void deleteItemData(String applicationId, String insertId, long versionNo)

指定された登録データ ID の登録情報を削除します。

◆ パラメータ
applicationId アプリケーション ID
insertId 登録データ ID
versionNo データバージョン番号
◆ 返却値
なし

後処理内でこのメソッドを使用する場合、さらに後処理を実行する必要がない場合はコンストラクタ②の execPostProcFlg を false にしてマネージャオブジェクトを生成してください。

6.1.2.5 登録情報マップ

登録情報マップは Map<String, Object>形式で登録情報を保持します。

- Map<String, Object>
  - 明細テーブル以外のアイテム
    キー:フィールド識別 ID 値: 入力値

  - 明細テーブルアイテムのデータ
    キーはテーブル識別ID、値は1レコード(行)毎にフィールド識別IDと値をマッピングした Map の List 各列のデータ型は列タイプに依存します。

※データの更新時は排他チェック用に更新対象データのバージョン番号が必須です。
登録情報マップにキー:imfr_sd_version_no でバージョン番号を設定してください。
※ファイルアップロードアイテムの情報は登録データに含まれません。

アイテムのデータ型と値のデータ型

アイテムのデータ型	値のデータ型
文字列	java.lang.String
数値	< データの登録・更新時 > java.lang.Number のサブクラス < データ取得時 > java.math.BigDecimal
日付またはタイムスタンプ	java.util.Date

6.2 スクリプト開発モデル

スクリプト開発モデルにおいて、後処理プログラムを実装する手順を示します。
アプリケーションデータを登録または検索・削除するための下記のオブジェクトが用意されています。

FRApplicationDataManager

6.2.1 コンストラクタ

■ コンストラクタ①

・ **FRApplicationDataManager(String loginUserCd, String loginGroupId)**

このコンストラクタを利用してマネージャオブジェクトを生成した場合、データ操作を行った際に後処理の設定があった場合は後処理が実行されます。

- パラメータ
  loginUserCd ログインユーザコード
  loginGroupId ログイングループID

■ コンストラクタ②

・ **FRApplicationDataManager(String loginUserCd, String loginGroupId, Boolean execPostProcFlg)**

このコンストラクタを利用してマネージャオブジェクトを生成時にデータ操作を行った際の後処理実行有無を指定することができます。

- パラメータ
  loginUserCd ログインユーザコード

loginGroupId　　ログイングループID
execPostProcFlg　後処理実行フラグ　後処理を実行する場合は true、しない場合は false

6.2.2 主なメソッド

6.2.2.1 データの取得メソッド

ItemDataObject getItemData (String applicationId, String insertId, Boolean systemData)

指定されたデータ登録 ID の登録情報を返却します。

- パラメータ
  applicationId アプリケーション ID
  insertId ログデータ登録 ID
  systemData　システム情報の取得の有無　取得する場合 true、しない場合は false
- 返却値
  登録情報オブジェクト※詳細は後述の「6.2.2.5 ItemDataObject 登録情報オブジェクト」をご参照ください。

6.2.2.2 データの登録メソッド

void insertItemData(String applicationId, Number applicationNo, String insertId, ItemDataObject object)

アプリケーションの登録情報をテーブルに登録します。

- パラメータ
  applicationId　アプリケーションID
  applicationNo アプリケーションバージョンNO
  insertId　　登録データID
  ItemDataObject　登録情報オブジェクト※詳細は後述の「6.2.2.5 ItemDataObject 登録情報オブジェクト」をご参照ください。
- 返却値
  なし

後処理内でこのメソッドを使用する場合、さらに後処理を実行する必要がない場合はコンストラクタ②の execPostProcFlg を false にしてマネージャオブジェクトを生成してください。

6.2.2.3 データの更新メソッド

void udpateItemData(String applicationId, Number applicationNo, String insertId, ItemDataObject object)

指定された登録データ ID の登録情報を更新します。

- パラメータ
  applicationId　アプリケーションID
  applicationNo アプリケーションバージョンNO
  insertId　　登録データID
  ItemDataObject　登録情報オブジェクト※詳細は後述の「6.2.2.5 ItemDataObject 登録情報オブジェクト」をご参照ください。
- 返却値
  なし

後処理内でこのメソッドを使用する場合、さらに後処理を実行する必要がない場合はコンストラクタ②の execPostProcFlg を false にしてマネージャオブジェクトを生成してください。

6.2.2.4 データの削除メソッド

void deleteItemData(String applicationId, String insertId, Number versionNo)

指定された登録データ ID の登録情報を削除します。

- ◆ パラメータ
  applicationId　アプリケーション ID
  insertId　登録データ ID
  versionNo　データバージョン番号
- ◆ 返却値
  なし

後処理内でこのメソッドを使用する場合、さらに後処理を実行する必要がない場合はコンストラクタ②の execPostProcFlg を false にしてマネージャオブジェクトを生成してください。

6.2.2.5 ItemDataObject 登録情報オブジェクト

- ■ 明細テーブル以外のアイテム
  プロパティ:フィールド識別 ID
- ■ 明細テーブルアイテムのデータ
  プロパティ:テーブル識別ID
  値は1レコード(行)毎に各列のフィールド識別IDと値をマッピングしたオブジェクトの配列です。
  各列のデータ型は列タイプに依存します。

※データの更新時は排他チェック用に更新対象データのバージョン番号が必須です。
登録情報オブジェクトの imfr_sd_version_no プロパティにバージョン番号を設定してください。
※ファイルアップロードアイテムの情報には登録データには含まれません。

- ■ アイテムのデータ型と値のデータ型の対比

アイテムのデータ型	値のデータ型
文字列	String
数値	Number
日付またはタイムスタンプ	Date

7 テンプレートの格納先

7.1 入力チェックプログラム

入力チェックプログラムのテンプレートは以下に格納されています。

%Resource Service%/pages/src/sample/forma/template/input_validation.js

7.2 後処理プログラム

7.2.1 JavaEE開発モデル

JavaEE 開発モデルの後処理プログラムのテンプレートは以下のページよりダウンロードしてください。

http://www.intra-mart.jp/download/product/v72_src/im_forma/ im_forma_sample-src.zip

7.2.2 スクリプト開発モデル

スクリプト開発モデルの後処理プログラムのテンプレートは以下に格納されています。

%Resource Service%/pages/src/sample/forma/template/post_process.js

IM-FormaDesigner Ver.7.2
プログラミングガイド

2012/10/26 第2版

TEL: 03-5549-2821
FAX: 03-5549-2816
E-MAIL: info@intra-mart.jp
URL: http://www.intra-mart.jp/