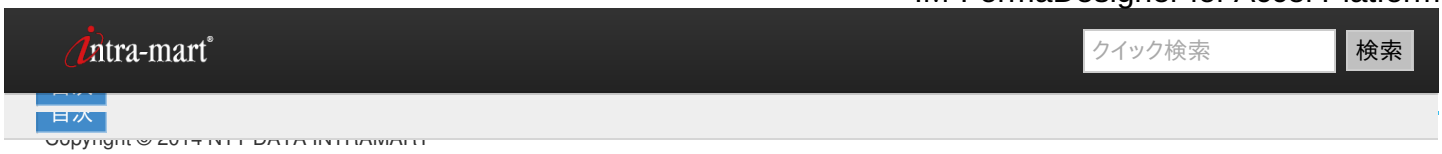


目次

改訂情報

変更年月日	変更内容
2014-12-24	初版
2015-04-01	第2版 下記を追加しました。 ■ 「*画面アイテムを「タブ切替」を設定したフォームで利用する場合の注意点* 」

はじめに

本書の目的

本書では、IM-FormaDesigner for Accel Platform（以降「 IM-FormaDesigner 」）の「フォーム・デザイナ」画面で利用する「画面アイテム」を作成するために必要な情報と手順を説明します。

- IM-FormaDesigner では、製品で用意している画面アイテム以外に、新規に画面アイテムを作成して利用することができます。

対象読者

本書は、IM-FormaDesigner の画面アイテムを新規作成・拡張を行う技術開発者を対象としております。
IM-FormaDesigner の基本機能、スクリプト開発モデル等に関する知識を習得していることを前提に説明しております。

IM-FormaDesigner の基本的な機能については、「 IM-FormaDesigner 作成者操作ガイド 」を参照ください。
スクリプト開発モデルに関する情報については、「 スクリプト開発モデル プログラミングガイド 」を参照してください。

本書の構成

- *概要*

  IM-FormaDesigner の画面アイテム作成の概要についてご理解いただけます。

- *画面アイテムの開発*

  IM-FormaDesigner の画面アイテム作成の手順の詳細について説明しております。

- *付録*

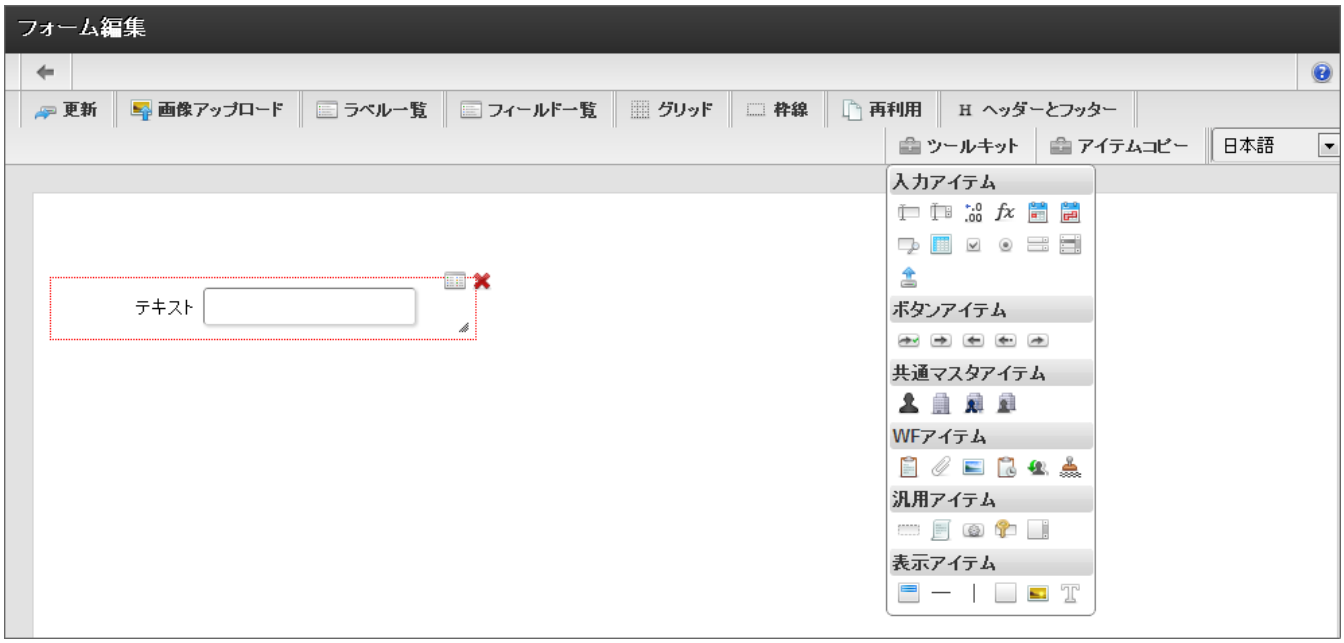
  IM-FormaDesigner で画面アイテム作成に役立つ補足情報をまとめております。

用語解説

用語名	説明
Webアーカイブディレクトリ	アプリケーションサーバ上でwarを展開したディレクトリを<%WAR%>と略します。
Storage	Storage として使用するディレクトリを<%STORAGE_PATH%>と略します。
Web Contents	Webコンテンツの静的コンテンツを配置するディレクトリを <%WEB_PATH%> と略します。

概要

画面アイテムとは、「フォーム・デザイナ」画面のツールキットからフォーム画面上に配置できる画面部品です。
ここでは、画面アイテムの開発の概要について説明します。



前提条件

- intra-mart e Builder for Accel Platform をインストール済みであること。
- intra-mart Accel Platform をインストールし、初期設定までが完了していること。
  アプリケーションにIM-FormaDesignerを含めて環境を作成してください。
  開発環境の Resin サーバは単体テスト用で作成してください。

画面アイテムを作成する基本的な流れ

画面アイテムを作成する基本的な流れは、以下の通りです。
詳細については「*画面アイテムの開発*」をご覧ください。

1. ユーザモジュールの作成と設定
2. 画面部品のUIや振舞いを決める
3. 「フォーム・デザイナ」画面上で設定可能なプロパティ項目を決める
4. プロパティ設定情報を保存するためのデータ構成を決める
5. 画面アイテムタイプを決め、画面アイテム構成ファイルを実装する
6. 実装した画面アイテムをツールキットに登録する

画面アイテムタイプと画面アイテム構成ファイル

画面アイテムタイプ

画面アイテムタイプとは、画面アイテムを識別するためのIDです。
イントラマートより提供している画面アイテムの画面アイテムタイプは必ず「product」から始まります。

そのため、独自に画面アイテムを作成する場合は「product」以外を使用してください。

画面アイテムを構成するファイル

画面アイテム構成ファイルとは、1つの画面アイテムを構成するファイル群です。
画面アイテム構成ファイルをそれぞれ実装し、ツールキットに作成した画面アイテムを登録すると、「フォーム・デザイナ」画面から作成した画面アイテムを利用できるようになります。

1つの画面アイテムを構成するファイルは、以下の通りです。

サーバサイドファイル（スクリプト開発モデルのファイル）

ファイル名	説明
type (js)	画面アイテム定義や入力チェックを実装するファイル
preview (html/js)	「フォーム・デザイナ」画面で表示されるプレビュー用の画面部品ファイル
properties (html/js)	画面アイテム情報を設定するプロパティ画面用の画面部品ファイル
input (html/js)	表示タイプ「入力可」の画面部品ファイル
reference (html/js)	表示タイプ「参照」の画面部品ファイル

クライアントサイドファイル（静的ファイル）

ファイル名	説明
%画面アイテムタイプ% (js)	画面アイテムのデータ構成定義を行うファイル

開発環境のセットアップ

プロジェクトの作成と設定

intra-mart e Builder for Accel Platform 上にモジュール・プロジェクトを作成し、プロジェクトの設定を行います。
プロジェクトの作成・設定の方法に関しては、「 intra-mart e Builder for Accel Platform ユーザ操作ガイド 」の「モジュール・プロジェクト作成」、及び「プロジェクトの設定」を参照してください。

モジュールの依存関係について

作成したモジュール・プロジェクトに「 IM-FormaDesigner 」モジュールへの依存関係を追加してください。
モジュールへの依存関係追加の方法に関しては、「 intra-mart e Builder for Accel Platform ユーザ操作ガイド 」の「module.xml」を参照してください。

サンプルプロジェクトのインポートと設定

サンプルの画面アイテムを含んだモジュール・プロジェクト「forma_sample_items」を **forma_sample_items.zip** より入手することができます。
このダウンロードした zip ファイルは、下記手順にて e Builder にインポートすることができ、すぐ実行確認することができます。
はじめに、下記手順にて e Builder にインポートしてください。

1. e Builder を起動
2. ツールバーメニュー[ファイル]-[インポート]よりインポートウィザード画面を開く
3. 項目[General]-[既存プロジェクトをワークスペースへ]を選択し次へ
4. [アーカイブ・ファイルの選択]項目よりダウンロードしたzipファイルを選択し、[終了]ボタンをクリック
5. 以上の手順で、モジュール・プロジェクト「forma_sample_items」がインポートされます。

次に、モジュール・プロジェクト「forma_sample_items」をデバッグサーバの環境に適用します。
適用することで、サンプルの画面アイテム「案件名」「コメント」が利用できます。

1. モジュール・プロジェクトのプロパティにてWebアーカイブディレクトリを設定します。
2. プロジェクトのクリーンを実行します。
3. クリーンの完了後に、デバッグサーバを起動します。
4. 「フォーム・デザイナ」画面のツールキットからサンプルの画面アイテム「案件名」「コメント」を利用することができます。

サンプルの画面アイテムは、アプリケーション種別(BIS作成種類)が「IM-Workflow」「BIS-ワークフロー」以外の場合は利用できません。

サンプルについて

サンプルの画面アイテムの仕様を簡単に紹介します。

- 画面アイテム「案件名」
  - テキストボックスに入力した値がIM-Workflowの案件名に連動します。
  - アプリケーション種別(BIS作成種類)の利用範囲は「IM-Workflow」「BIS-ワークフロー」となります。
- 画面アイテム「コメント」
  - テキストエリアに入力した値がIM-Workflowのコメント欄に連動します。
  - アプリケーション種別(BIS作成種類)の利用範囲は「IM-Workflow」「BIS-ワークフロー」となります。

> **注意**
>
> サンプルの画面アイテムは、IM-FormaDesigner for Accel Platform 8.0.8-PATCH_001が適用された環境にて検証しております。

画面アイテムの設計

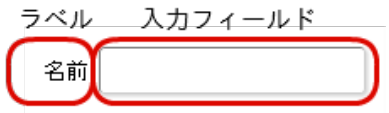
画面部品のUIや振舞いを決める

画面アイテムの作成の最初の手順として、画面アイテムのUI（見た目）や振舞いを決定する必要があります。

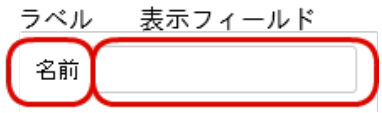
画面アイテムのUIと振舞い

はじめに、どのような画面アイテムを作成するかを決めます。
具体的には、その画面アイテムのUI と振舞いを決めます。
UI と振舞いは、2つの表示タイプ「入力可」と「参照」それぞれで定義します。
たとえば、画面アイテム「文字列」のUI では、「入力可」の場合はラベルと入力フィールドがそれぞれ1つずつ配置され、「参照」の場合はラベルと登録されたデータを表示するためのフィールドが配置しています。

- 表示タイプ「入力可」



- 表示タイプ「参照」



画面アイテムのUIを決定する

UI としては、以下の3タイプがあります。

1. ユーザが入力したデータを登録する項目を持つ画面アイテムの場合、入力フィールドを持つ必要があります。
   【例】画面アイテム「文字列」、「数値」
2. 1つの画面アイテムで複数の項目を入力させたい場合は、1つの画面アイテムの中に複数の入力フィールドを持ちます。
   【例】画面アイテム「期間」
3. 表示のみを目的とする場合は、1つの画面アイテムに1つも入力項目を持ちません。
   【例】画面アイテム「見出し」

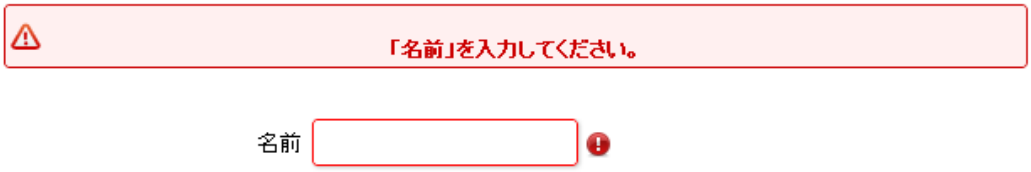
画面アイテムの振舞いを決定する

次に、振舞いを決定します。
振舞いには入力チェックやイベント処理が該当します。
入力フィールドがある場合は、登録時の入力チェックやフィールドがフォーカスされたときやフォーカスが外れたときの制御、画面アイテム内にボタンがある場合は、ボタンが押下された時の動作などの各種イベントを決定します。

画面アイテム「文字列」では、以下のような入力チェックが可能です。

1. 必須チェック
2. 文字数のチェック（最大値と最小値）
3. 文字の種類のチェック（半角英数字）

入力チェックでエラーが発生した場合は、下図のように、入力フィールドに対してエラーがわかり易いように枠の色の変更と、エラーのアイコンを表示するようにしています。



画面アイテム「数値」では、プロパティを設定することで、フィールドがフォーカスされたときとフォーカスが外れたときに以下のような処理が可能です。

- フォーカスされたとき、「3桁カンマ区切り表示」から「数値のみ表示」に変更
- フォーカスが外れたとき、「数値のみ表示」から「3桁カンマ区切り表示」に変更
  - フォーカスされていない状態

- フォーカスされている状態

画面アイテム「一覧選択」では、アイコンがクリックされたときに以下のように一覧画面を表示します。

このように、画面アイテムのUIと振舞いを決めます。

「フォーム・デザイナ」画面上で設定可能なプロパティを決める

画面アイテムで設定可能なプロパティ項目を決めます。
一部のプロパティ項目は、「フォーム・デザイナ」画面上で設定可能な項目です。
(「フォーム・デザイナ」画面上で設定できないプロパティも存在します。例:自動的にランダムな値で設定されるアイテムID)
プロパティ項目には、あらかじめ基本プロパティが用意されています。
基本プロパティに関しては「*基本プロパティ項目一覧*」をご覧ください。
新しく画面アイテムを作成する場合、基本プロパティにない画面アイテム独自の必要なプロパティを整理し、プロパティ画面でどのように設定できるようにするか、プロパティ画面のUIを決めてください。

プロパティ画面

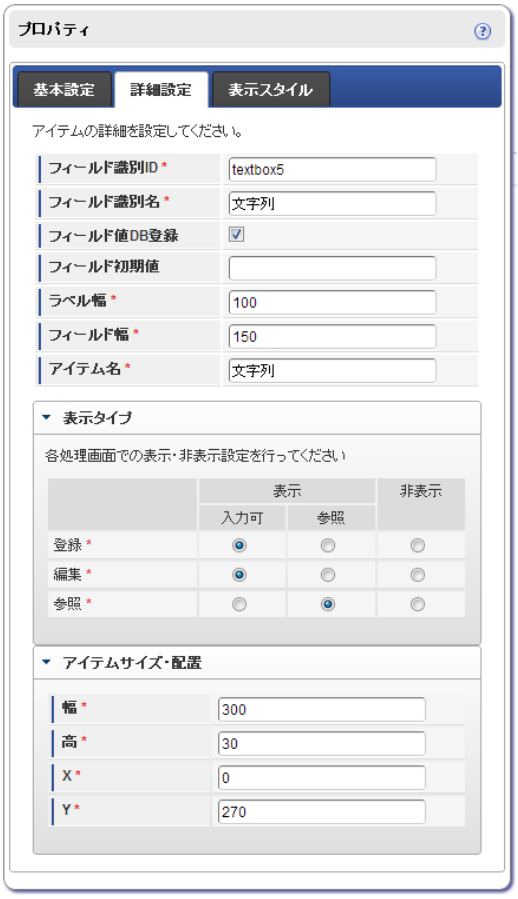
画面アイテム「文字列」の場合、「フォーム・デザイナ」画面上のプロパティ画面は以下になります。

- 基本設定

設定項目	プロパティ名
ラベル	labels
必須入力チェック	required
英数字のみ	only_ascii
最小入力文字数	min_length
最大入力文字数	max_length

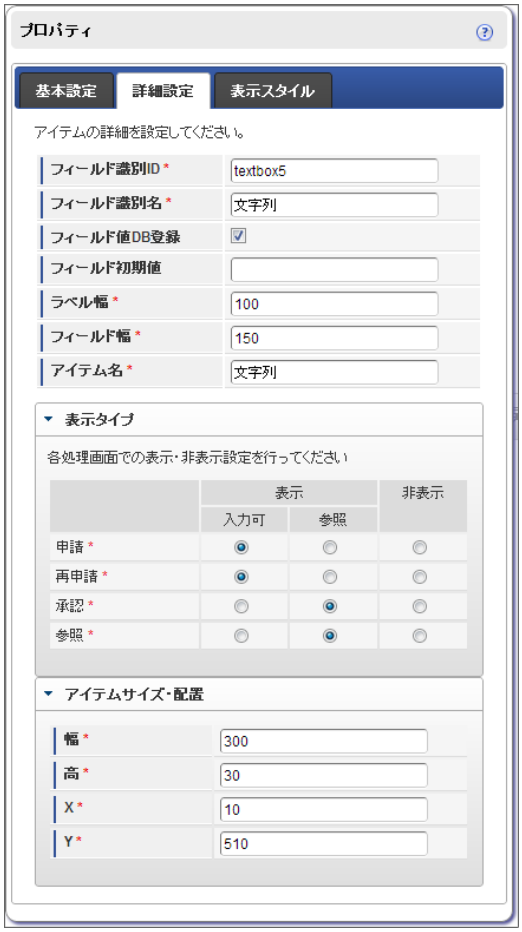
- 詳細設定(標準アプリケーションの場合)

  表示タイプの設定内容は、登録、編集、参照単位に配列で登録されます。

	設定項目	プロパティ名
	■ フィールド識別 ID ■ フィールド識別名 ■ フィールド値 DB 登録 ■ フィールド初期値 ■ ラベル幅 ■ フィールド幅 ■ アイテム名 ■ 表示タイプ ■ 幅 ■ 高 ■ X ■ Y	■ input_id ■ input_view_names ■ item_exist_dbinput ■ initial_value ■ label_size ■ input_field_size ■ item_view_names ■ item_view_type ■ width ■ height ■ left ■ top

- 詳細設定（ワークフローアプリケーションの場合）

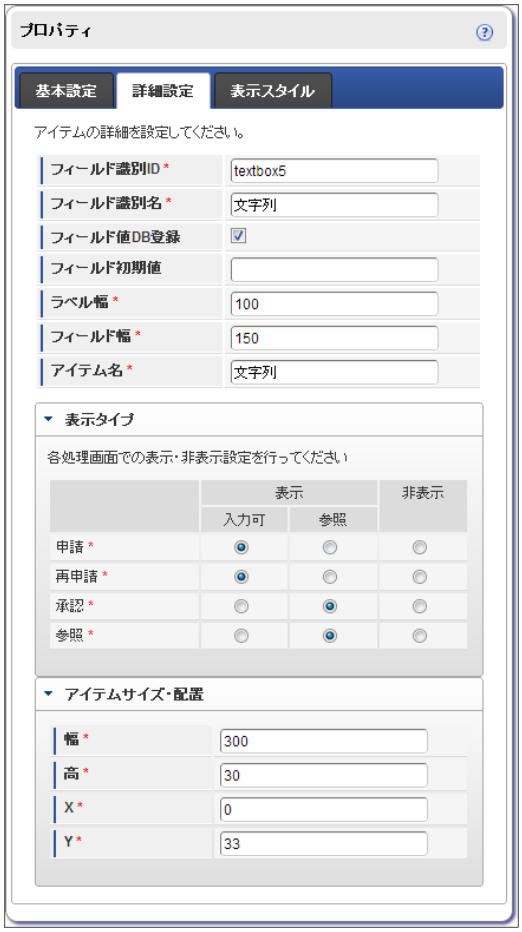
  表示タイプの設定内容は、申請、再申請、承認、参照単位に配列で登録されます。



設定項目	プロパティ名
フィールド識別 ID	input_id
フィールド識別名	input_view_names
フィールド値 DB 登録	item_exist_dbinput
フィールド初期値	initial_value
ラベル幅	label_size
フィールド幅	input_field_size
アイテム名	item_view_names
表示タイプ	item_view_type
幅	width
高	height
X	left
Y	top

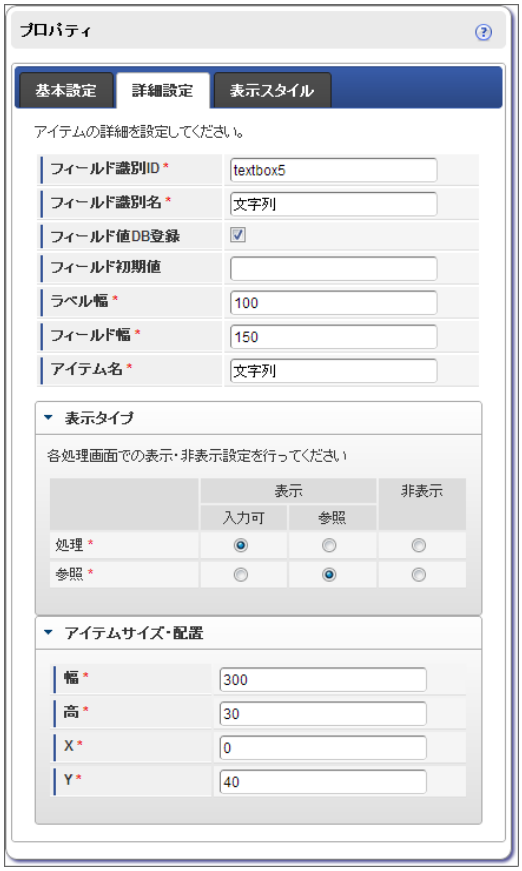
- 詳細設定（BIS-ワークフローの場合）

  表示タイプの設定内容は、申請、再申請、承認、参照単位に配列で登録されます。

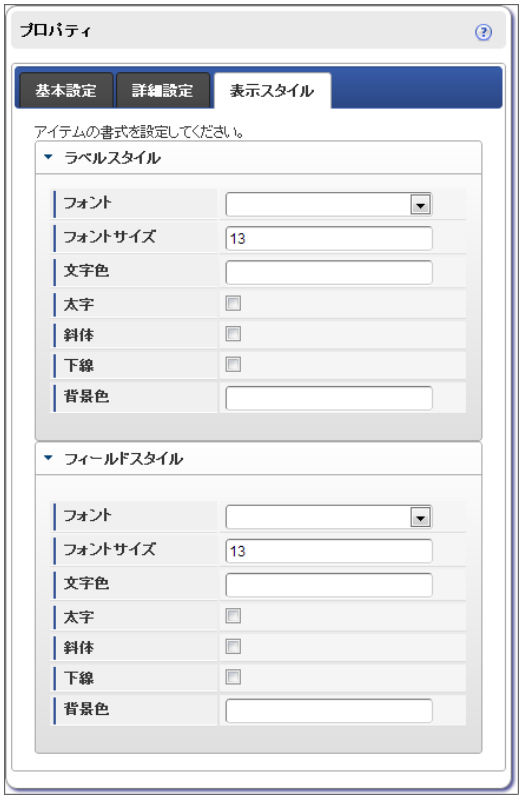
	設定項目	プロパティ名
	■ フィールド識別 ID ■ フィールド識別名 ■ フィールド値 DB 登録 ■ フィールド初期値 ■ ラベル幅 ■ フィールド幅 ■ アイテム名 ■ 表示タイプ ■ 幅 ■ 高 ■ X ■ Y	■ input_id ■ input_view_names ■ item_exist_dbinput ■ initial_value ■ label_size ■ input_field_size ■ item_view_names ■ item_view_type ■ width ■ height ■ left ■ top

- 詳細設定（BIS-BPMの場合）

  表示タイプの設定内容は、処理、参照単位に配列で登録されます。

	設定項目	プロパティ名
	■ フィールド識別 ID ■ フィールド識別名 ■ フィールド値 DB 登録 ■ フィールド初期値 ■ ラベル幅 ■ フィールド幅 ■ アイテム名 ■ 表示タイプ ■ 幅 ■ 高 ■ X ■ Y	■ input_id ■ input_view_names ■ item_exist_dbinput ■ initial_value ■ label_size ■ input_field_size ■ item_view_names ■ item_view_type ■ width ■ height ■ left ■ top

- 表示スタイル

	設定項目	プロパティ名
	▪ ラベルスタイル ▪ フォント ▪ フォントサイズ ▪ 文字色 ▪ 太字 ▪ 下線 ▪ 背景色 ▪ フィールドスタイル ▪ フォント ▪ フォントサイズ ▪ 文字色 ▪ 太字 ▪ 下線 ▪ 背景色	▪ label_styles ▪ label_font_family ▪ label_font_size ▪ label_font_color ▪ label_font_bold ▪ label_font_underline ▪ label_font_italic ▪ input_properties ▪ input_font_family ▪ input_font_size ▪ input_font_color ▪ input_font_bold ▪ input_font_underline ▪ input_font_italic

基本プロパティ項目一覧

基本プロパティ項目の一覧は以下の通りです。

item_id	アイテムID	▪ フォーム内の画面アイテムで一意な、画面アイテム識別コードです。 ▪ 自動的に設定されます。
item_view_names	アイテム名	▪ 画面アイテムの表示名です。ワークフローの追記設定などで使用されます。
item_type	アイテムタイプ	▪ 画面アイテムの識別IDです。画面アイテム専用CSJSファイルで定義します。
item_view_type	表示タイプ	▪ 画面アイテムの登録、参照など各画面での表示タイプです。
item_exist_dbinput	DB登録値存在フラグ	▪ 画面アイテム内に、データベースへと登録する項目が存在するかどうかのフラグです。 ▪ 1画面アイテム内に、1つでも登録項目が存在する場合はTRUEを設定します。
style	表示スタイル	▪ 画面アイテムの表示に関する項目を定義します。

top	Y	▪ 画面アイテムの左上を基点に、フォーム上端からの表示位置を設定します。
left	X	▪ 画面アイテムの左上を基点に、フォーム左端からの表示位置を設定します。
width	幅	▪ 画面アイテムの表示幅を設定します。
height	高さ	▪ 画面アイテムの表示高さを設定します。
item_properties	アイテム詳細設定	▪ 画面アイテムに属する詳細設定を定義します。
input_id	フィールド識別ID	▪ 入力フィールドを識別するIDです。 ▪ フォーム内で一意である必要があります。 ▪ データベースのテーブルカラム名や、登録済みデータを受け取る時のプロパティ名になります。 ▪ 「フォーム・デザイナ」画面のフィールド一覧機能を使用する時に自動的に取得・編集されます。
input_data_type	フィールドデータ型	▪ 入力フィールドのデータ型を設定します。
input_view_names	フィールド識別名	▪ 入力フィールド名を設定します。登録データー覧画面の項目名の他、データベースのカラム論理名に使用されます。
input_dbinput	フィールド値DB登録	▪ 入力フィールドをデータベースへ登録するかどうかを設定します。登録する場合は値をtrueにします。
input_list_display	一覧表示設定可否	▪ 入力フィールドを一覧表示設定画面で設定できるかどうかを設定します。設定できるようにする場合は値をtrueにします。また、一覧表示設定画面で設定できるようにするためには、input_dbinputがtrueである必要があります。この項目を省略した場合はtrueとして動作します。
input_properties	入力フィールド詳細設定	▪ 入力フィールドに属する詳細設定を定義します。
input_field_size	フィールド幅	▪ 入力フィールドの横幅を設定します。
labels	フィールドラベル	▪ 入力フィールドの入力補助ラベルを設定します。 ▪ この項目の設定値は、画面アイテムの入力補助ラベルと違い、ラベルー覧機能では自動的に取得・編集されません。
initial_value	初期値	▪ 入力フィールドの初期値を設定します。
tab_index	タブインデックス	▪ フォーム上でTabキーを押下した場合の、入力フィールドのカーソル遷移順を設定します。
max_length	最大入力文字数	▪ ユーザ入力チェック時の入力フィールドの最大入力文字数を設定します。
min_length	最小入力文字数	▪ ユーザ入力チェック時の入力フィールドの最小入力文字数を設定します。
required	必須入力チェック	▪ ユーザ入力チェック時の必須入力チェックの有無を設定します。

利用可能なフィールドデータ型

プロパティで利用可能なフィールドデータ型は4つあり、それぞれに該当する値、プログラムの定数と、データベースにおけるカラムのデータ型は以下のようになります。

データ型	プログラム定数と値	カラムデータ型 (初期値)
文字列	LIBRARY.dataTypeString = “0”	IMFR_DATA_TYPE_STRING (VARCHAR)

データ型	プログラム定数と値	カラムデータ型 (初期値)
数値	LIBRARY.dataTypeNumber = “1”	IMFR_DATA_TYPE_NUMBER (DECIMAL)
日付	LIBRARY.dataTypeDate = “2”	IMFR_DATA_TYPE_DATE (DATE)
日時	LIBRARY.dataTypeTimestamp = “9”	IMFR_DATA_TYPE_TIMESTAMP (TIMESTAMP)

プログラムの定数は、共通ライブラリファイルitems_library.js で定義されています。
各定数に一致するデータベースにおけるカラムデータ型は、以下のように<%WAR%>/WEB-INF/conf/forma-config.xml で設定します。

```
:(省略)

<!--  データ型 文字列 -->
<data_type_string>varchar</data_type_string>
<!-- データ型 数値 -->
<data_type_number>decimal</data_type_number>
<!-- データ型 日付 -->
<data_type_date>date</data_type_date>
<!-- データ型 タイムスタンプ -->
<data_type_timestamp>timestamp</data_type_timestamp>

:(省略)
```

コラム

カラムデータ型は、使用するデータベースに応じて正しく設定する必要があります。
詳細は「 IM-FormaDesigner セットアップガイド 」をご覧ください。

プロパティ設定情報を保存するためのデータ構成を決める

プロパティ項目を決定したら、次にプロパティ項目に合わせてデータ構造を決定します。

画面アイテムのプロパティのデータ構造

前述のプロパティのデータは、下記のようなツリー構造になっています。

- itemObject（画面アイテムプロパティ）
  - item_id （アイテムID）
  - item_view_names （アイテム名）
  - item_type （アイテムタイプ）
  - item_view_type （表示タイプ）
  - item_exist_dbinput （DB登録値存在フラグ）
  - style （表示スタイル）
    - top （Y）
    - left （X）
    - width （幅）
    - height （高さ）
  - item_properties （アイテム詳細設定）
    - labels （ラベル）
    - label_size （ラベル幅）
    - label_styles （ラベルスタイル）
      - label_background_color （背景色）
      - label_font_color （文字色）
      - label_font_size （フォントサイズ）
      - label_font_family （フォント）
      - label_font_bold （太字）
      - label_font_italic （斜体）
      - label_font_underline （下線）
  - input_list （入力フィールドリスト）
    - input_id （フィールド識別ID）
    - input_data_type （データ型）
    - input_view_names （フィールド識別名）
    - input_dbinput （フィールド値DB登録）
    - input_list_display （一覧表示設定可否）
    - input_properties （入力フィールド詳細設定）
      - input_field_size （フィールド幅）
      - labels （フィールドラベル）
      - initial_value （初期値）
      - tab_index （タブインデックス）
      - max_length （最大入力文字数）
      - min_length （最小入力文字数）
      - required （必須入力チェック）
      - input_background_color （背景色）
      - input_font_color （文字色）
      - input_font_size （フォントサイズ）
      - input_font_family （フォント）
      - input_font_bold （太字）
      - input_font_italic （斜体）
      - input_font_underline （下線）

プロパティ項目を追加する場合は、内容に合わせてデータ構成の「アイテム詳細設定」内、もしくは、「入力フィールドリスト」の下の「入力フィールド詳細設定」内に追加してください。

画面アイテム「文字列」の場合、設定内容は以下のとおりです。値はJSON形式で記述しています。設定する箇所は、次章にて解説します。

「フォーム・デザイナ」画面のラベル一覧機能

ラベル一覧は、フォーム上に配置されている画面アイテムのラベルを一覧で表示、変更する機能です。
画面アイテムからの値の取得、変更後値の設定は「フォーム・デザイナ」画面側から行われ、画面アイテム側で何らかの関数を実装、実行する必要はありません。
「フォーム・デザイナ」画面で自動的にアイテムラベルの有無、設定値の取得を行いますが、取得元のデータ構成は以下のように構成する必要があります。

labels には、ロケールID をプロパティキー、表示ラベルを値としたオブジェクトを、存在するラベルの数分の配列として設定します。

- 例

```
labels = [
  {
    ja : '日本語ラベル1',
    en : 'english label1'
  },
  {
    ja : '日本語ラベル2',
    en : 'english label2'
  }
];
```

「フォーム・デザイナ」画面のフィールド一覧機能

フィールド一覧は、フォーム上に配置されている入力項目を一覧で表示、変更する機能です。
変更可能なものは、フィールド識別名とタブインデックスになります。
画面アイテムからの値の取得、変更後値の設定は「フォーム・デザイナ」画面側から行われます。
itemObject のinput_listに保持している全入力項目をフィールド一覧に表示する場合は、画面アイテム側で処理を実装する必要はありません。
フィールド一覧に表示する入力項目を指定する場合は、%画面アイテムタイプ% (js)でgetFieldList関数を実装する必要があります。
詳細は後述の「*%画面アイテムタイプ% (js)*」をご参照ください。

「フォーム・デザイナ」画面で自動的に入力項目の有無、設定値の取得を行いますが、取得元のデータ構成は以下のように構成する必要があります。

input_view_names には、ロケールID をプロパティキー、フィールド識別名を値としたオブジェクトを設定します。

フィールドデータ型「日付」「日時」を利用する場合に表示フォーマットが必須

フィールドデータ型が「日付」および「日時」のデータは、1970 年1 月1 日午前0 時からの経過時間（ミリ秒数）で扱います。
このデータを画面上に表示する場合は、表示フォーマットを利用します。

そのため、フィールドデータ型が「日付」および「日時」の場合、以下の表示フォーマットを定義するプロパティ項目date_format が必要です。

一覧表示設定をさせたくない入力項目の設定

ユーザアプリ一覧に表示させる項目の設定は、一覧表示設定画面で行います。
基本的に、プロパティ項目input_dbinput がtrue の入力項目だけが、一覧表示設定画面で設定できます。
画面アイテム内にユーザアプリ一覧に表示させたくない入力項目がある場合、その入力項目にプロパティ項目input_list_display を追加しfalse に設定します。
こうすることにより、DB 登録する入力項目(プロパティ項目input_dbinput = true)でも一覧表示設定画面に表示させないことができます。
プロパティ項目 input_list_display のない入力項目、または、プロパティ項目input_list_display がtrue の入力項目は、プロパティ項目input_dbinput がtrue の場合、一覧表示設定画面で設定することができます。

画面アイテムタイプの決定

画面アイテムタイプとは、画面アイテムを識別するためのID です。
画面アイテム構成ファイルとは、1つの画面アイテムを構成するファイル群です。

画面アイテムタイプと画面アイテム構成ファイルが格納されるフォルダには関連があります。
たとえば、画面アイテム「文字列」の場合、画面アイテムタイプは「product_72_textbox」です。

このとき画面アイテムの構成ファイルのサーバサイドファイルは、「<%WAR%>/WEB-INF/jssp/product/src/forma/designer/types/product/72/textbox/」フォルダに格納されます。
クライアントサイドファイルは、「<%WEB_PATH%>/forma/csjs/types/product/72/textbox.js」となります。

> **注意**
>
> 画面アイテムタイプでは、アンダーバーをフォルダの区切りとして解釈するため、<%WEB_PATH%>/forma/csjs/types以下の格納フォルダにおいて、**フォルダ名の中に「アンダーバー」文字を利用できません。**
> サーバサイド側は上記の制約はありませんが、同じルールに従ってディレクトリを作成することをおすすめします。

> **注意**
>
> 製品より提供している画面アイテムの画面アイテムタイプは、必ず「product」から始まります。
> そのため独自に画面アイテムを作成する場合は、「product」以外を使用してください。

画面アイテム構成ファイルの実装

画面アイテムタイプを決定したら、次は画面アイテム構成ファイルを実装します。1つの画面アイテムを構成するファイルは、以下の通りです。

- サーバサイドファイル（ スクリプト開発モデル のファイル）

ファイル名	説明
type (js)	画面アイテム定義や入力チェックを実装するファイル
preview (html/js)	「フォーム・デザイナ」画面で表示されるプレビュー用の画面部品ファイル
properties (html/js)	画面アイテム情報を設定するプロパティ画面用の画面部品ファイル
input (html/js)	表示タイプ「入力可」の画面部品ファイル
reference (html/js)	表示タイプ「参照」の画面部品ファイル

- クライアントサイドファイル（静的ファイル）

ファイル名	説明
%画面アイテムタイプ% (js)	画面アイテムのデータ構成定義を行うファイル

上記のように、画面アイテム構成ファイルには、スクリプト開発モデルのプレゼンテーション・ページ、ファンクション・コンテナと静的ファイルであるクライアントサイドJavaScript（以下 CSJS）で構成されます。
各画面アイテム構成ファイルの役割は大きく2つに分かれます。

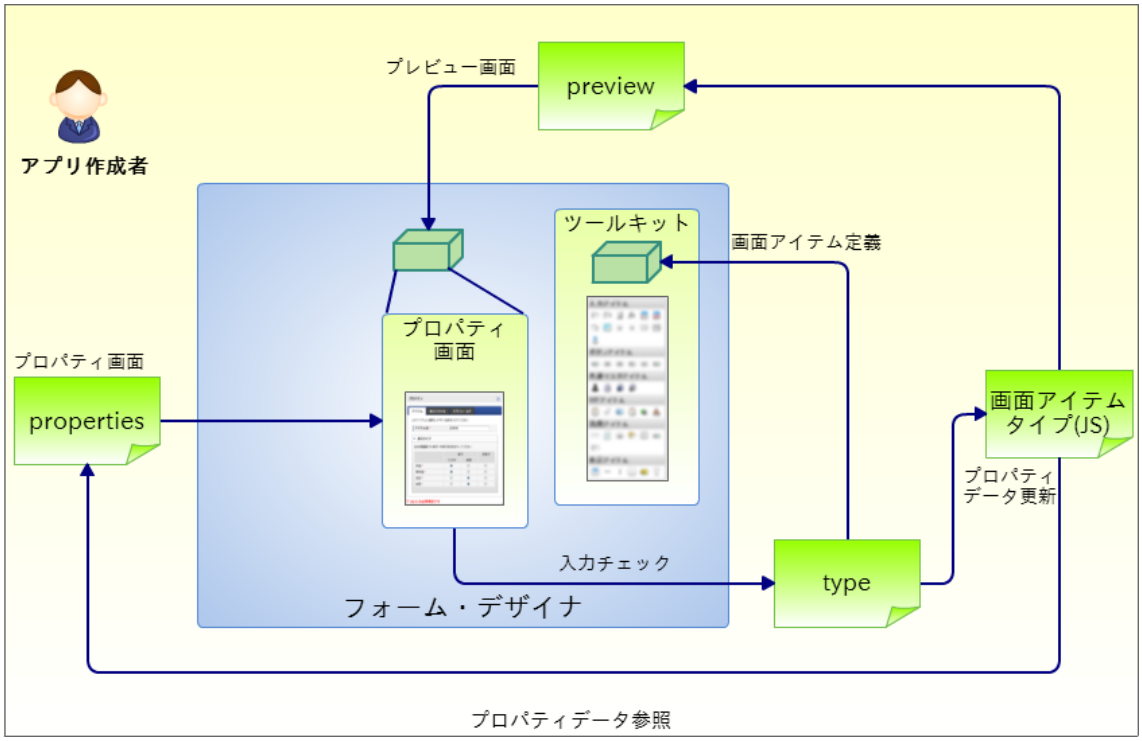
- アプリ作成者が利用する「フォーム・デザイナ」画面のためのファイル
  - %画面アイテムタイプ%(js)
  - preview(html/js)
  - properties(html/js)
  - type(js)
- アプリ利用者が利用する登録・編集画面/参照画面のためのファイル
  - input(html/js)
  - reference(html/js)
  - type(js)

以下、「*アプリ作成者が利用する「フォーム・デザイナ」画面のためのファイル* 」と「*アプリ利用者が利用する登録・編集画面/参照画面のためのファイル*」について各ファイルの関連を説明します。
画面アイテム構成ファイルを実装するために利用可能な共通関数については、「 *サーバサイド共通ライブラリ・ファイル* 」をご覧ください。
構成ファイルの詳細については、「*画面アイテム構成ファイル*」をご覧ください。

アプリ作成者が利用する「フォーム・デザイナ」画面のためのファイル

アプリ作成者が利用する「フォーム・デザイナ」画面のためのファイルの関連は以下のイメージの通りです。

- %画面アイテムタイプ%(js)
- preview(html/js)
- properties(html/js)
- type(js)



以下では、「*プロパティ変更処理のシーケンス*」と「*プロパティ入力エラー処理のシーケンス*」について説明します。

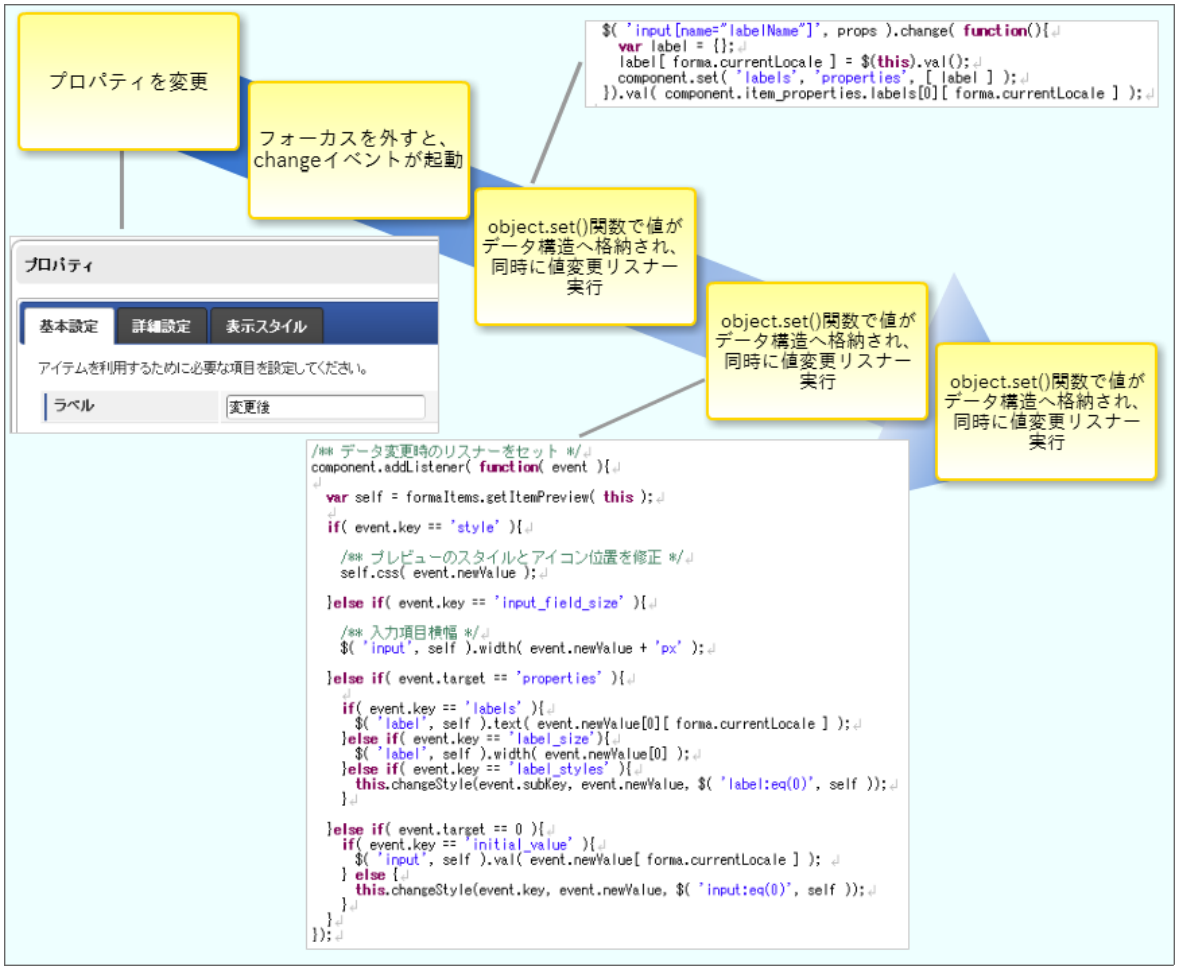
プロパティ変更処理のシーケンス

アプリ作成者がプロパティ画面(properties)で入力した内容が確定されたときに、即時に「フォーム・デザイナ」画面上のプレビュー画面(preview)に反映する処理シーケンスについて説明します。
プロパティ画面で入力された値をプレビュー画面に反映する処理イメージは以下の通りです。

- プロパティ画面とプレビュー画面 (変更前)

- プロパティ画面とプレビュー画面 (変更後)

プロパティ項目からカーソルを移動させると、チェンジイベントが発生し、properties.html に記述されていた項目値のデータ構造へのセットリスナーが実行されます。
このチェンジイベントは、jQuery のchange を利用しています。
続いて、値の変更が行われたため preview.html に記述されていた値変更リスナーが実行されます。
値変更リスナーには、変更後の値にプレビュー上の表示値を置き換える処理が記述されていますので、プレビューのラベル表示値が変更されます。

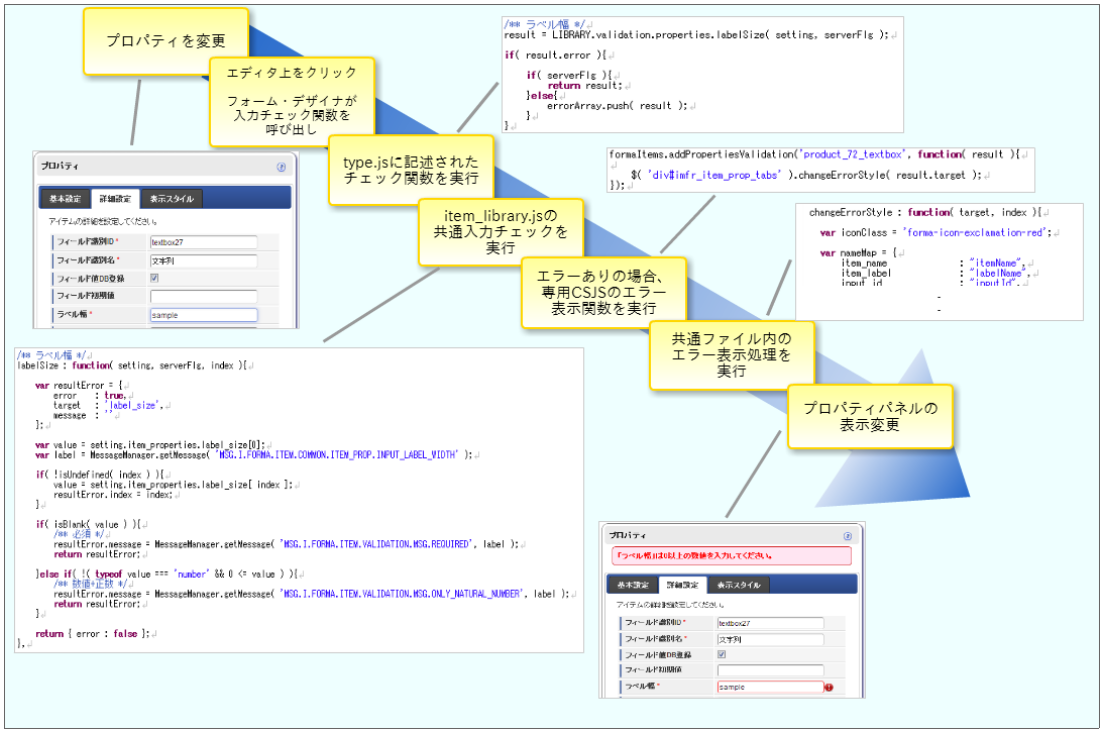


プロパティ入力エラー処理のシーケンス

アプリ作成者がプロパティ画面(properties)で入力した内容が確定されたときに、その入力内容のチェックを行います。
その入力チェック処理シーケンスについて説明します。
プロパティ画面で入力された値をチェックし、エラーの場合は以下の画面イメージのようにプロパティ画面の上部にエラー内容を表示します。

- プロパティ項目の入力エラー表示

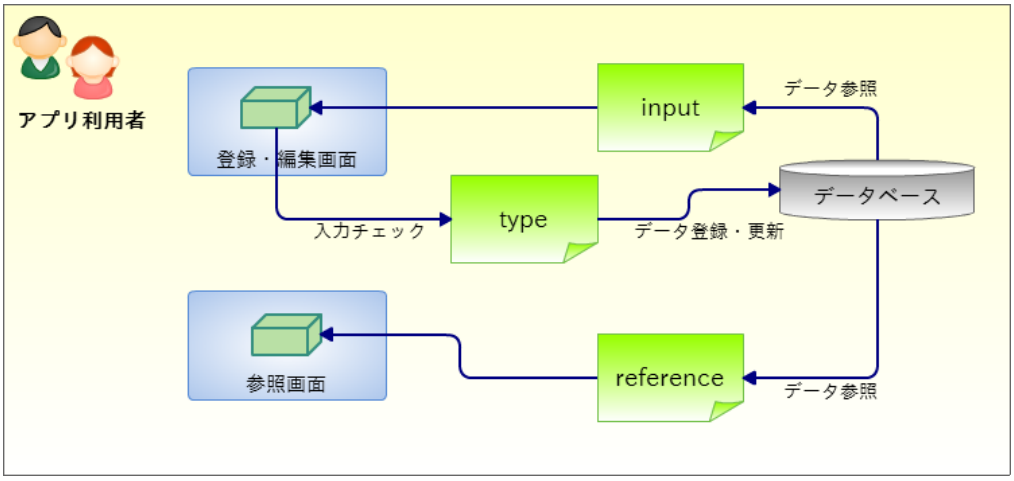
このように、プロパティ画面で入力された値をチェックし、エラーの場合にプロパティ画面にエラー内容を表示する処理シーケンスは以下の通りです。



アプリ利用者が利用する登録・編集画面/参照画面のためのファイル

アプリ利用者が利用する登録・編集画面/参照画面の各ファイルの関連は以下のイメージの通りです。

- input(html/js)
- reference(html/js)
- type(js)



以下では、「*表示タイプ*」と「*プロパティ入力エラー処理のシーケンス*」について説明します。

表示タイプ

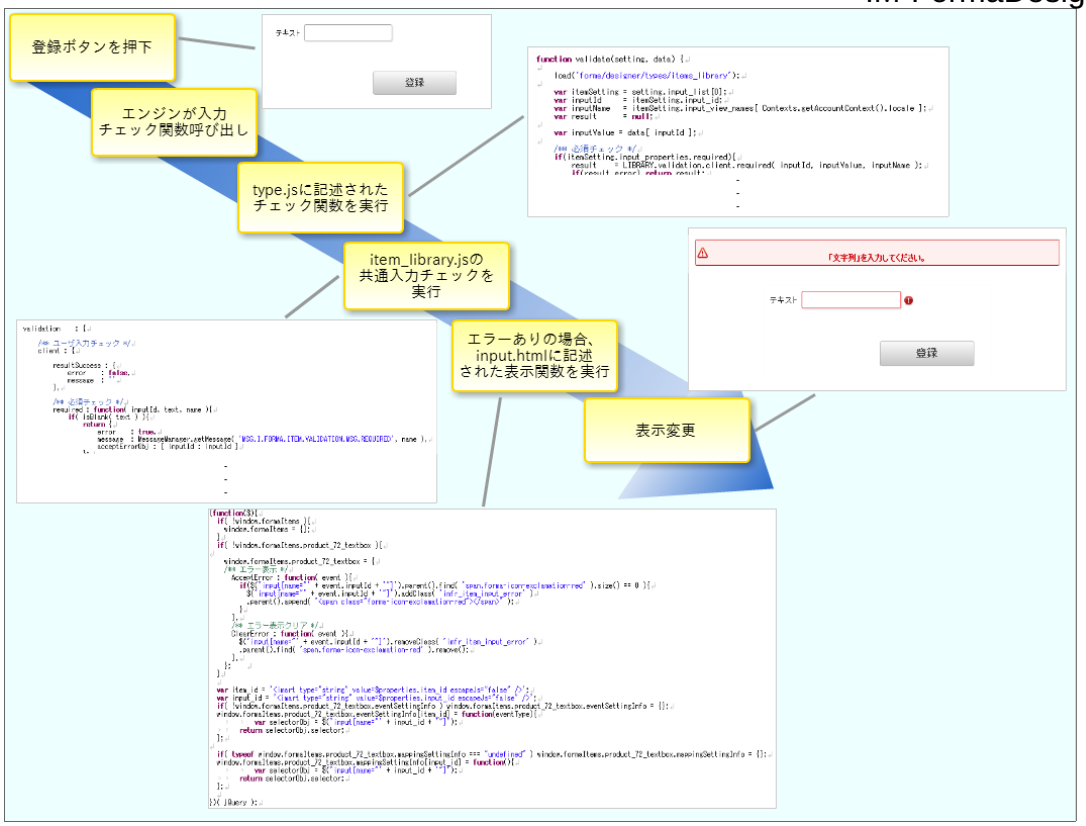
「フォーム・デザイナ」画面上の画面アイテムのプロパティ画面で設定された表示タイプを元に、各処理画面で表示するファイルが決定されます。

表示タイプ	使用ソース
入力可	input (.html/.js)
参照可	reference (.html/.js)
表示	reference (.html/.js)
非表示	何も表示しない

プロパティ入力エラー処理のシーケンス

アプリ利用者が登録・編集画面で入力した値は、登録/更新ボタンをクリックしたときにチェックを行います。
その入力チェック処理シーケンスについて説明します。
入力エラーの場合は、以下の画面イメージのように登録・編集画面の上部にエラー内容を表示します。

このように、登録・編集画面で入力された値をチェックし、エラーの場合に画面上部にエラー内容を表示する処理シーケンスは以下の通りです。



画面アイテムのタブインデックス

アプリ利用者の登録・編集画面での[TAB]キーが押されたときにフォーカスが移動する順番は、フォーム作成時にフィールド一覧のタブインデックスで設定することができます。
入力項目のある画面アイテムは、フォーム・データをもとに対象入力項目にHTML タグのtabindex 属性を指定する必要があります。
画面アイテム作成時には、どのようにタブインデックスを設定させ、割り当てていくかを考慮して作成してください。

標準で提供する入力項目のある画面アイテムのタブインデックスには以下のようなパターンがあります。

1. 1入力項目に対し1インデックスを持つパターン

   【例】文字列、数値

   【実装】

   - 入力項目にtabindex 属性を指定する。
2. 固定の複数入力項目に対し入力項目数分インデックスを持つパターン

   【例】期間

   【実装】

   - 各入力項目にそれぞれのtabindex 属性を指定する。
3. 可変の複数入力項目に対し単一インデックスを持つパターン

   【例】チェックボックス、ラジオボタン

   【実装】

   - 各入力項目に単一のtabindex 属性を指定する。
   - %画面アイテムタイプ% (js)でgetFieldList 関数を実装し、フィールド一覧に表示する入力フィールドを指定する。
     詳細は後述の「*%画面アイテムタイプ% (js)*」を参照してください。
4. 可変の複数入力項目に対し複数インデックスを持つパターン

   【例】明細テーブル

   【実装】

   - 各入力項目にそれぞれのtabindex 属性を指定する。

- 登録画面で複数の入力項目が動的に追加されるような場合は、必要に応じて入力項目追加時にフォーム内の要素の tabindex の再割り当て処理を実装する。

警告・インフォメーションメッセージの表示

登録・編集画面/参照画面を表示する際、画面アイテムが表示するべき値を取得できなかった場合などに画面アイテムが指定した警告またはインフォメーションメッセージを画面上部に表示することができます。

- 画面アイテム「ユーザ選択」の警告メッセージを表示している例

  - 実装方法
    input.html またはreference.html で以下の関数を利用して各メッセージを登録します。
    - forma.util.addWarning
      警告メッセージ登録関数
      引数：メッセージ文字列またはメッセージ文字列の配列
    - forma.util.addInfomation
      インフォメーションメッセージ登録関数
      引数：メッセージ文字列またはメッセージ文字列の配列

各メッセージの表示処理は$(document).ready で行われるため、$(document).ready が実行される前に表示するメッセージを登録してください。
メッセージにXML エスケープが必要な文字が含まれる可能性がある場合は、エスケープ後のメッセージ文字列を設定してください。

- 実装例

```
<imart type="condition" validity=$data.error>
  forma.util.addWarning( '<imart type="string" value=$data.errorMessage />' );
</imart>
```

実装した画面アイテムをシステムに登録する

ツールキットへの登録

実装した画面アイテムを「フォーム・デザイナ」画面で画面アイテムとして使用できるよう、設定ファイルに登録します。

画面アイテムの登録は、以下の設定ファイルで行います。

```
<%WAR%>/WEB-INF/conf/forma-extension-config/forma-config_xxx.xml
```

> **コラム**
>
> 配置するファイル名は、IM-BIS で提供する「forma-config_bis.xml」と重複しないようにしてください。
> 上記の条件に合致していれば、ファイル名は任意の名前を設定できます。

> **コラム**
>
> 画面アイテムに対するヘルプの設定については、「 IM-FormaDesigner 作成者操作ガイド 」の「IM-FormaDesigner の高度な設定を行う」を参照してください。

1. 画面アイテム構成ファイルの格納フォルダのパスを以下の場所に登録します。

```
…
 <toolkit-setting>
    <define-item>
       <item-path>%画面アイテム構成ファイルの格納フォルダのパス%</item-path>
    …
```

   - IM側で提供している画面アイテム構成ファイルの格納フォルダのパスの先頭は、forma/designer/types/になります。
     たとえば、画面アイテム「文字列」の場合は、以下のようになります。
     新規に画面アイテムを作成する場合はproductフォルダ以下は使用せずに、新規にフォルダを作成してください。

```
…
 <toolkit-setting>
    <define-item>
       <item-path>forma/designer/types/product/72/textbox</item-path>
    …
```

2. 次に、画面アイテムをツールキット上に表示させるための設定を行います。
   既存のグループに追加する場合は、追加したいグループの子に、画面アイテムタイプを設定します。

```
…
 <toolkit-setting>
    …
    <default-grouping>
       <group>
          <group-id>%グループID%</group-id>
          <item-id>%画面アイテムタイプ%</item-id>
    …
```

   - 画面アイテム「文字列」の場合は、以下のようになります。

```
…
 <toolkit-setting>
    …
    <default-grouping>
      <group>
        <group-id>input</group-id>
        <item-id>Product_72_textbox</item-id>
    …
```

3. 次に、画面アイテムをどのアプリケーション種別（BIS作成種類）で表示するかを設定します。
   設定しない場合には、すべてのアプリケーション種別（BIS作成種類）のツールキットで表示されます。
   複数のアプリケーション種別（BIS作成種類）を指定する場合には、カンマ区切りで設定します。

```
…
 <toolkit-setting>
    …
    <default-grouping>
      <group>
        <group-id>%グループID%</group-id>
        <item-id application-type="{対象のアプリケーション種別（BIS作成種類）}">%画面アイテムタイプ%</item-id>
    …
```

   - 画面アイテム「ボタン（登録）」の場合は、以下のようになります。

```
…
 <toolkit-setting>
    …
    <default-grouping>
      <group>
        <group-id>button</group-id>
        <item-id application-type="std,imw,bis_wkf">product_80_registButton</item-id>
    …
```

4. ツールキットで画面アイテムを表すアイコンを設定します。
   アイコンとして利用するCSSスプライト用の画像とCSSファイルのパスを指定します。

```
…
 <toolkit-setting>
    …
    <css-path>{CSSファイルパス}</css-path>
    …
```

   - IM-BIS の画面アイテムの場合は、以下のようになります。

```
…
 <toolkit-setting>
    …
    <css-path>forma/css/bis-forma-ui-icons.css</css-path>
    …
```

CSS Spriteの追加

CSS Spriteとは複数の画像部品を連結して1枚の画像ファイルにまとめ、CSSで表示範囲を指定することによって画像を表示する手法のことです。intra-mart Accel Platform における 画像表示にはCSS Sprite が利用されています。

1. Sprite作成ツールのセットアップを行います。intra-mart Accel Platformが提供するCSS Sprite作成ツールを利用します。**『CSS Sprite作成ツール』**をダウンロードし、解凍します。解凍されたファイル群をモジュール・プロジェクトのルート・ディレクトリ上にコピーします。
2. 以下の設定ファイル を開きます。

build_sprite.properties

3. artifact_id属性に対してモジュールのアーティファクトID（プロジェクト名）を指定します。
4. CSS Spriteを生成します。Sprite化したい画像ファイル群をsrc/main/public_not_compiledへコピーします。
5. 以下の設定ファイルを指定してAntを実行します。

build_sprite.xml

6. 2の処理の完了後、src/main/public/${artifact_id}/cssにcssファイルが生成されます。また、src/main/public/${artifact_id}/imagesに連結された画像ファイルが生成されます。
7. Spriteを設定ファイルに追加します。以下の設定ファイルを開きます。『toolkit-setting/css-path』の項目にsrc/main/publicからの相対パスを指定します。

forma-extension-config_${artifact_id}.xml

8. css-path属性にsrc/main/publicからの相対パスを指定します。

```
…
 <toolkit-setting>
    …
    <css-path>{CSSファイルパス}</css-path>
    …
```

画面アイテム・グループの追加方法

ツールキットに画面アイテム・グループを追加する方法は以下の通りです。

1. default-grouping の下へ、他のgroup-idと重複しないIDを持つgroupを作成します。

```
…
 <toolkit-setting>
    …
    <default-grouping>
      <group>
        <group-id>newgroup</group-id>
      </group>
    …
```

2. 次にメッセージプロパティファイルへグループ名を追加し、<%WAR%>/WEB-INF/jssp/product/src/forma/designer/form_designer_toolkit.js へ表示名を追加します。
画面アイテムタイプをキーとして、値にメッセージマネージャから取得した表示名を設定します。

```
var $groups;

function init(arg) {
 messages = {
   input:MessageManager.getMessage("imfr.form_designer.label.item_group.input"),
   button:MessageManager.getMessage("imfr.form_designer.label.item_group.button"),
   master:MessageManager.getMessage("imfr.form_designer.label.item_group.master"),
   workflow:MessageManager.getMessage("imfr.form_designer.label.item_group.workflow"),
   general:MessageManager.getMessage("imfr.form_designer.label.item_group.general"),
   display:MessageManager.getMessage("imfr.form_designer.label.item_group.display").
   newgroup: MessageManager.getMessage("imfr.form_designer.label.item_group.newgroup") // 追加グループ表示名
 }
 $groups = arg.groups;
 for(var i = 0; i < $groups.length; i++) {
    $groups[i].title = messages[$groups[i].id] ? messages[$groups[i].id] : "";
 }
}
```

3. 以上で、新規グループが作成されます。追加したグループに画面アイテムを追加してください。

画面アイテム・グループを特定のアプリケーション種別のときのみ使用可能にする方法

作成した画面アイテム・グループを特定のアプリケーション種別の時のみツールキットに表示してグループ内の画面アイテムを使用できるようにしたい場合は、上記で作成したgroup にapplication-type 属性を追加します。

```
<group application-type="std">
  <group-id> newgroup </group-id>
  .....
```

指定する値は以下の通りです。

- アプリケーション種別 標準・・・std
- アプリケーション種別 IM-Workflow・・・imw

IM-BIS のフローの場合は以下の通りです。

- BIS作成種類 BPM・・・bis_bf
- BIS作成種類 ワークフロー・・・bis_wkf

application-type 属性の指定がない画面アイテム・グループは、アプリケーション種別問わず常にツールキットに表示されます。

作成した画面アイテムを画面アイテム「一覧選択」の取得値フィールド対象外にする

画面アイテム「一覧選択」では、クエリで取得される項目が1 つの場合は、必ず自分自身のテキストボックスへ反映されますが、複数存在する場合は2 つ以降は他の画面アイテムの入力項目へ反映させることができます。
この時フォーム内に存在する入力フィールドリスト（input_list）を持つ画面アイテムが取得値を反映させる対象フィールドとなりますが、入力フィールドリストを持っていても作成した画面アイテムを画面アイテム「一覧選択」の取得値を反映させる対象フィールドにはしたくない場合は、以下の設定を行います。

この設定を行うことで、対象の画面アイテムのフィールドは取得値設定対象から除外され画面アイテム「一覧選択」の取得値設定セレクトボックスに表示されなくなります。

- 設定方法
  item-setting タグ内のexclude-itemselect 要素を追加し画面アイテムタイプを設定します。
  標準で提供する画面アイテムではチェックボックスやラジオボタンなどが設定されています。

```
<item-setting>
  <date-format>yyyy/MM/dd</date-format>
  .....
  <exclude-itemselect>product_72_checkbox</exclude-itemselect>
  <exclude-itemselect>product_72_listbox</exclude-itemselect>
  .....
  <exclude-itemselect>%画面アイテムタイプ%</exclude-itemselect>
</item-setting>
```

ツールキット設定ファイルの実装例

ツールキットの設定ファイルの実装例です。

サンプル1

以下のサンプルでは、2つの画面アイテムを次の内容で設定しています。

- 画面アイテム「hoge」
  - アイテムグループを「共通マスタ」に設定
  - アプリケーション種別（BIS作成種類）の利用範囲は「標準」のみ
- 画面アイテム「hoge2」
  - アイテムグループを「表示アイテム」に設定

- アプリケーション種別（BIS作成種類）の利用範囲は「すべて」（設定なし）

```
<?xml version="1.0" encoding="UTF-8"?>
<forma-extension-config xmlns="http://www.intra-mart.jp/forma-extension-config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.intra-mart.jp/forma-extension-config ../../schema/forma-extension-config.xsd">

  <toolkit-setting>
    <define-item>
      <item-path>forma/designer/types/product/sample/hoge</item-path>
      <item-path>forma/designer/types/product/sample/hoge2</item-path>
    </define-item>
    <default-grouping>
      <group>
        <group-id>master</group-id>
        <item-id application-type="std">hoge</item-id>
      </group>
      <group>
        <group-id>display</group-id>
        <item-id>hoge2</item-id>
      </group>
    </default-grouping>
  </toolkit-setting>
</forma-extension-config>
```

サンプル2

以下のサンプルでは、2つの画面アイテムを次の内容で設定しています。

- 画面アイテム「hoge」
  - アイテムグループを「ボタンアイテム」に設定
  - アプリケーション種別（BIS作成種類）の利用範囲は「標準」「IM-Workflow」「BPM」
- 画面アイテム「hoge2」
  - アイテムグループを「汎用アイテム」に設定
  - アプリケーション種別（BIS作成種類）の利用範囲は「BPM」「ワークフロー」
- アイコン用にCSSスプライトを追加

```
<?xml version="1.0" encoding="UTF-8"?>
<forma-extension-config xmlns="http://www.intra-mart.jp/forma-extension-config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.intra-mart.jp/forma-extension-config ../../schema/forma-extension-config.xsd">

  <toolkit-setting>
    <define-item>
      <item-path>forma/designer/types/product/sample/bis_hoge</item-path>
      <item-path>forma/designer/types/product/sample/bis_hoge2</item-path>
    </define-item>
    <default-grouping>
      <group>
        <group-id>button</group-id>
        <item-id application-type="std,imw,bis_bf">bis_hoge</item-id>
      </group>
      <group>
        <group-id>general</group-id>
        <item-id application-type="bis_bf,bis_wkf">bis_hoge2</item-id>
      </group>
    </default-grouping>
    <css-path>forma/css/bis-hoge.css</css-path>
  </toolkit-setting>
</forma-extension-config>
```

リンク機能のある画面アイテムを作成する

Contents

画面アイテム「一覧選択」のように、独自の画面を表示する画面アイテムを作成する場合には、以下の2つの対応が必要となります。

- 認可設定
- クライアントタイプ切替

認可設定について

画面アイテム側で独自に定義した画面とアプリケーションの画面とは同一のアクセス権が設定される必要があります。intra-mart Accel Platform上で同一のアクセス権を実現するには、ルーティングテーブルに対して同一の認可リソースとして登録する必要があります。

1. ルーティングテーブルの設定ファイルを作成します。

```
src/main/conf/routing-jssp-config/${artifact_id}.xml
```

2. 画面のパスに対して登録画面用と編集,参照画面用の2つURLを作成します。

```
<file-mapping path="%登録画面用のURL%/{applicationId}" page="%画面ファイルのパス%">
<file-mapping path="%編集,参照画面用のURL%/{applicationId}" page="%画面ファイルのパス%">
```

3. 登録画面のURLにアプリケーションの登録画面用の認可リソースを紐づける。

```
<file-mapping path="%登録画面用のURL%/{applicationId}" page="%画面ファイルのパス%">
  <authz uri="forma-service://forma/regist_application_view/{applicationId}" action="execute" />
</file-mapping>
```

4. 編集、参照画面のURLにアプリケーションの編集,参照画面用の認可リソースを紐づける。

```
<file-mapping path="%編集,参照画面用のURL%/{applicationId}" page="%画面ファイルのパス%">
  <authz uri="forma-service://forma/list_view/{applicationId}" action="execute" />
</file-mapping>
```

クライアントタイプ切替

アプリケーションの画面はPCテーマ向けに作られており、スマートフォン版からアクセスする場合も当該リクエストのみPCテーマに切り替えて表示しています。
画面アイテムの独自画面を表示する際にも同様に、テーマの切替処理が必要となります。

- 独自画面のファンクション・コンテナ内に以下の処理を記述してください。

```
if (Procedure.imfr_view_utils.isSmartphoneClientType()) {
                ClientTypeSwitcher.oneTimeSwitchTo('pc');
}
```

注意

認可、ルーティングについては、それぞれ「 スクリプト開発モデル プログラミングガイド 」の「認可」、「ルーティング」を参照してください。

注意

クライアントタイプについては、「 スクリプト開発モデル プログラミングガイド 」の「UI(スマートフォン開発ガイドライン) クラ

イアントタイプとテーマ」を参照してください。

注意

クライアントタイプ切替処理については、「 intra-mart Accel Platform スクリプト開発向けim-BizAPI 」の「ClientTypeSwitcherオブジェクト」を参照してください。

共通ライブラリ・ファイル

サーバサイド共通ライブラリ・ファイル

items_library.js

ファイル名	ディレクトリ	概要
items_library.js	<%WAR%>/WEB-INF/jssp/product/src/forma/designer/types	サーバサイドでの共通仕様定数と共通処理を記述しているファイルです。 load 関数でファイルを読み込んで使用します。

- 関数
  - LIBRARY.validation.properties
    「フォーム・デザイナ」画面上で、画面アイテムのプロパティ画面から入力した時に行われる入力チェックです。
    各画面アイテムで共通なプロパティ項目は、ここに記述されている関数を利用して入力チェックを行なっています。
    主なチェック関数名と対象プロパティは以下のとおりです。

チェック関数名	対象プロパティ
itemViewNames	アイテム名チェック
labelName	ラベル名チェック
inputId	フィールド識別ID チェック
labelSize	ラベル幅チェック
itemViewType	アイテム表示タイプチェック
inputViewNames	フィールド識別名チェック
inputFieldSize	入力フィールド横幅チェック
valueLength	最小・最大入力文字数チェック
numberValue	最小・最大入力値チェック
styles	アイテムの表示スタイルチェック

  - LIBRARY.validation.client
    「フォーム・デザイナ」画面で作成したフォームを使用してデータを登録する時に行われるユーザ入力チェック用の共通関数です。
    各画面アイテムで共通なチェック項目は、ここに記述されている関数を使用して入力チェックを行なっています。
    主なチェック関数名は以下のとおりです。

チェック関数名	対象プロパティ
required	必須入力チェック
selectRequired	必須選択チェック
minLength	最小文字数チェック
maxLength	最大文字数チェック
onlyAscii	英数字のみチェック
onlyNumber	数値のみチェック
minusNumber	負数チェック
numberSize	数値のサイズ（最小・最大値）チェック
decimalSize	小数部桁数チェック
dateFormat	日付形式チェック

チェック関数名	対象プロパティ
required	添付ファイルの個数チェック

- 定数
  - LIBRARY.dataTypeString
    登録データ形式（文字列）の画面アイテムプロパティ設定値が定義されています。
  - LIBRARY.dataTypeNumber
    登録データ形式（数値）の画面アイテムプロパティ設定値が定義されています。
  - LIBRARY.dataTypeDate
    登録データ形式（日付）の画面アイテムプロパティ設定値が定義されています。
  - LIBRARY.dataTypeTimestamp
    登録データ形式（タイムスタンプ）の画面アイテムプロパティ設定値が定義されています。
  - LIBRARY.viewType
    表示タイプの画面アイテムプロパティ設定値が定義されています。
  - LIBRARY.appType
    アプリケーションタイプが定義されています。
    画面アイテム内でアプリケーションタイプを判定する必要がある場合に使用します。
  - LIBRARY.dateFormats
    設定ファイルで定義されている、日付形式を配列で取得します。

クライアントサイド共通ライブラリ・ファイル

itemObject.js

ファイル名	ディレクトリ	概要
itemObject.js	<%WEB_PATH%>/forma/csjs/types	画面アイテムの基本データ構造と各種リスナー、画面アイテム用ユーティリティ関数が記述されています。 画面アイテムの基礎となる関数等を提供しています。そのため本ファイルを編集する場合は、影響範囲を考慮の上、編集を行ってください。

- 画面アイテム基礎系関数
  - FormaItemObject
    関数全ての画面アイテムに共通な、基本データ構造を定義しています。
    画面アイテムをフォームに新しく配置する場合は、new 演算子を使用してこの関数の新しいオブジェクトを生成し、1 画面アイテムに対して1 オブジェクトを持ちます。
    生成処理は「フォーム・デザイナ」画面側で自動的に行われるため画面アイテム側で行う必要はありません。
  - FormaItemObject.set 関数
    生成した画面アイテムオブジェクトへ、プロパティで入力や変更された値をセットする時に使用する関数です。
    主にproperties(.html/.js)で使用します。
    この関数を使用して値のセットを行なうと、後述のリスナー追加関数を利用してオブジェクトに追加されたリスナーを自動的に実行します。
    アイテムオブジェクトへ値をセットする場合は必ずこの関数を使用するようにして下さい。
    引数と使用方法は以下のとおりです。

object.set(key, target, value)

key [string]　セットするデータのプロパティ名を指定します。

target [string|number]　画面アイテム構造上のどこのプロパティへセットするかを指定します。

itemobject

- %properties%　←ここへセットする場合は[null]を指定
- item_properties
- %properties%　←ここへセットする場合は[properties]を指定
- input_list
  - %properties%　←ここへセットする場合は[input_list の添字]を指定
  - input_properties
    - %properties%　←ここへセットする場合は[input_properties の添字]を指定

value [object]　セットする値を指定します。

- FormaItemObject.addListener 関数
  画面アイテム構造へデータをセットした時に実行されるリスナー関数を設定します。
  主に preview(.html/.js)で使用します。
  引数で与える関数の this は画面アイテム構造自身を指し、引数event に与えられているプロパティと例文は以下のとおりです。

object.addListener(function(event){})

oldValue [object]　変更前セット値

newValue [object]　変更後セット値

key [string]　セットするデータのプロパティ名

target [string|number]　画面アイテム構造上のどこのプロパティへセットするか

  - 使用例

```
object.addListener( function( event ){
  if( event.key == 'foo' ){
      alert( 'old:' + event.oldValue + ',new:' + event.newValue );
  }
} );
```

- FormaItemObject.addLocaleListener 関数
  「フォーム・デザイナ」画面の表示ロケールが変更された時に実行されるリスナー関数を設定します。
  引数で与える関数の this は画面アイテム構造自身を指し、引数locale には変更後のロケールID が与えられます。

object.addLocaleListener(function(locale){})

locale [string] 変更後ロケールID

- FormaItemObject.addItemDef 関数
  画面アイテムのデータ構造や、初期化時の関数など画面アイテム定義情報を設定します。
  主に、クライアントサイドの画面アイテム専用js で使用します。
  引数は以下のとおりです。

```
object.addItemDef( type, model, initFunc )
type [string]　アイテムタイプ
model [object]　アイテム個別データ構造
initFunc [function]　データ構造オブジェクト生成時に実行される初期化関数
```

- FormaItemObject.addItemPreviewDef 関数
  ラベル初期値などの多言語情報が必要なプロパティなどの、追加分の画面アイテムのデータ構造と、画面アイテムプレビューの初期化を行なう関数を設定します。
  引数は以下のとおりです。

```
object.addItemPreviewDef( type, initPreview, initLocaleData )
type [string]　アイテムタイプ
initPreview [function]　プレビュー初期化関数
initLocaleData [function]　返却値のobject が、追加画面アイテム構造としてクライアントjs で設定されたデータ構造とマージされます
```

- ユーティリティ系関数
  - formaItems.getItemPreview 関数
    配置時に生成した画面アイテム毎のデータ構造オブジェクトから、フォーム上の画面アイテムプレビューのdivElement を取得します。

```
formaItems.getItemPreview( object )
object [object]　アイテムデータ構造
```

  - formaItems.addPropertiesValidation 関数
    プロパティの入力チェックエラーハンドリング関数を設定します。
    引数は以下のとおりです。

```
formaItems.addPropertiesValidation( type, func )
type [string]　アイテムタイプ
func [function]　エラーハンドリング関数：引数としてエラー内容result が与えられます。
```

  - jQuery 拡張changeErrorStyle 関数
    プロパティ画面入力時にエラーが発生した場合のエラーハンドリングを記述します。
    各画面アイテム個別の処理は、%画面アイテムタイプ% (js)で記述するので、ここでは共通なもののみを記述します。
    通常のテキストフィールドでの入力の場合は、関数内の nameMap 変数へ、入力チェック関数から返却されるtarget 名と、テキストボックスのname 属性値を追加することで、自動的にテキストボックスのスタイルが変更されます。

```
jQuery.fn.extend({ changeErrorStyle : function( target, index ) })
target [string]　エラー発生プロパティ項目のキー
index [number]　同名のプロパティが複数存在する場合のインデックス（任意）
```

jquery.formEditor.js

ファイル名	ディレクトリ	概要
jquery.formEditor.js	<%WEB_PATH%>/forma/csjs	「フォーム・デザイナ」画面用 CSJS ファイルです。 画面アイテムでも利用可能なユーティリティ関数が存在します。

- forma.current 変数
  現在選択（編集）中画面アイテムの、プレビューdiv のelement と、画面アイテムタイプが保存されています。
- forma.getComponent 関数
  forma.current 変数を引数として与えると、現在選択（編集）中画面アイテムの、データ構造を返却します。

データ構造が存在しない場合は null を返却します。

- forma.getComponentById 関数
  アイテム ID から、画面アイテムのデータ構造を返却します。
  データ構造が存在しない場合は null を返却します。

画面アイテム構成ファイル

サーバサイド・ファイル

type (js)

ファイル名	ディレクトリ	概要
type.js	forma/designer/types/ 画面アイテムタイプ	各画面アイテム固有のプロパティ情報データ構成定義と画面アイテムプロパティ入力チェック、ユーザ入力時の入力チェックを記述します。

- init 関数
  init 関数の返却値として「フォーム・デザイナ」画面に読み込まれる画面アイテム定義情報を記述します。
  返却されるオブジェクト情報を「フォーム・デザイナ」画面が受け取り、ツールキットに表示します。
  このファイルのロードタイミングはサーバの起動時です。変更の際はサーバの再起動を行なってください。

```
返却 object の各プロパティ
id      アイテムID
icon    ツールキットで表示されるアイテムアイコン
title   ツールキットで表示されるアイテム名のメッセージプロパティキー
preload   初期ロード設定：false にしてください
defaultStyle   画面アイテムをツールキットからドラッグして配置した時点の初期サイズ
   ├─height    アイテム高さ
   └─width     アイテム幅
```

画面アイテム「文字列」の場合

```
返却 object の各プロパティ
id      'product_72_textbox'
icon    'forma-icon-ui-text-field'
title   'MSG.I.FORMA.ITEM.TEXTBOX.TITLE'
preload   false
defaultStyle   画面アイテムをツールキットからドラッグして配置した時点の初期サイズ
   ├─height    '30px'
   └─width     '300px'
```

- validate 関数
  ユーザが入力した値の入力チェックを記述します。
  「フォーム・デザイナ」画面で作成されたフォームの登録処理が行われた場合に、自動的に実行されます。
  関数名は固定です。変更した場合、自動的に実行はされません。
  引数として以下の変数が与えられます。

```
validate( setting, data )
setting     画面アイテムデータ構造
data    フォーム全体の入力値
```

画面アイテム「文字列」の場合、使用入力チェック関数は以下のとおりです。

required	必須入力チェック
minLength	最小文字数チェック
maxLength	最大文字数チェック
onlyAscii	英数字のみチェック

- propertiesValidate 関数
「フォーム・デザイナ」画面上で、画面アイテムプロパティに入力された値の入力チェックを記述します。
画面アイテム個別のチェック時は、プロパティ画面が閉じられる時に実行されます。
関数名は固定です。変更した場合、自動的に実行はされません。
引数として以下の変数が与えられます。

```
propertiesValidate( setting, serverFlg )
setting　　画面アイテムデータ構造
serverFlg　チェック処理実行タイミング （現時点ではfalse のみ）
```

画面アイテム「文字列」の場合、使用入力チェック関数は以下のとおりです。

itemViewNames	アイテム名チェック
labelName	ラベル名チェック
labelSize	ラベル幅チェック
inputId	フィールド識別ID チェック
itemViewType	アイテム表示タイプチェック
inputViewNames	フィールド識別名チェック
inputFieldSize	入力フィールド横幅チェック
valueLength	最小・最大入力文字数チェック
lengthIntegrity	最小・最大文字数の整合性チェック
styles	アイテムの表示スタイルチェック

- getHeaderSettingList 関数
ユーザ入力時にロードする必要のある外部ファイルを記述します。
ロードしたいファイルの種類によって、決められた形式のオブジェクトを返却します。
ロードファイルが複数ある場合は、配列にして返却、ロードする必要がない場合は空オブジェクトを返却します。
オブジェクトのプロパティと例文は以下のとおりです。

```
type [string]　ファイルタイプ(javascript|jssprpc|stylesheet)
src [string]　ファイルパス(javascript の場合のみ)
href [string]　ファイルパス(stylesheet の場合のみ)
media [string]　スタイル適用メディアタイプ(stylesheet の場合のみ)
```

  - 使用例

```
function getHeaderSettingList(){
 return [
    {
        type : 'javascript',
        src : 'forma/csjs/types/xxx/xxx.js'
    },
    {
        type : 'stylesheet',
        href : 'forma/css/types/xxx/xxx.css',
        media : 'print'
    }
 ];
}
```

画面アイテム「文字列」の場合、ロードするファイルはありませんので空のオブジェクトを返しています。

preview (html/js)

ファイル名	ディレクトリ	概要
preview (.html/.js)	forma/designer/types/画面アイテムタイプ	「フォーム・デザイナ」画面上で表示される、編集中フォームの画面アイテムプレビューと、画面アイテムプレビューの定義を記述します。 formaItems.addItemPreviewDef 関数をここで記述することにより、初めてプレビューが表示された時に、画面アイテム別基本データ構造の初期化が行われます。 また、画面アイテムプレビューの初期化処理を記述することによって、プロパティで変更された値のプレビューへの即時反映（画面アイテムプロパティ値変更リスナー）や、「フォーム・デザイナ」画面の表示ロケール変更時のプレビュー表示ロケール変更（ロケール変更リスナー）などが行えます。

- formaItems.addItemPreviewDef 関数
  preview.html で必ず記述する必要があります。
  関数の詳細は、クライアントサイド共通ファイルのitemObjectを参照して下さい。
  画面アイテム「文字列」の場合、関数の処理は以下のとおりです。
  - 画面初期値を表示する処理
  - プロパティ画面によるデータ変更時の処理
  - ロケール変更時に、指定したロケールに対応する処理

properties (html/js)

ファイル名	ディレクトリ	概要
properties (.html/.js)	forma/designer/types/画面アイテムタイプ	画面アイテムプロパティを入力するための入力フィールドを記述します。 記述するHTML構造はある程度決まっており、下記の形式で記述することによって、自動的にタブ化などの処理が行われます。 プロパティの各項目が変更された際に、変更内容をプレビュー画面に動的に反映させるための処理も記述します。

```
<div id='imfr_item_prop_error_mes' class='state-error'></div>
 <div id='imfr_item_prop_tabs'>
   <ul>
     <li><a href='#imfr_item_prop_tab1'>xxx</a></li>
     <li><a href='#imfr_item_prop_tab2'>xxx</a></li>
     <li><a href='#imfr_item_prop_tab3'>xxx</a></li>
   </ul>
   <div id='imfr_item_prop_tabs1' class='imfr_item_prop_body'>
     タブ1 の内容
   </div>
   <div id='imfr_item_prop_tabs2' class='imfr_item_prop_body'>
     タブ2 の内容
   </div>
   <div id='imfr_item_prop_tabs3' class='imfr_item_prop_body'>
     タブ3 の内容
   </div>
</div>
<div id='imfr_item_help' class='ui-corner-all' ></div>
```

上記が基本的な形式になります。

id 属性値やCSS クラス名は固定ですので、上記のとおりに記述して下さい。
id=imfr_item_prop_error_mes のdiv は入力エラーメッセージの表示エリアになります。
プロパティ入力チェックが行われた後、エラーがある場合は自動的にここにメッセージが表示されます。
id= imfr_item_help のdiv は、アイテムプロパティヘルプを表示するためのエリアになります。
ヘルプ内容は別ファイルへ記述しますので、この部分に直接ヘルプ内容を書かないで下さい。
プロパティの値が変更された時に値を設定する関数は必ずcomponent.set 関数を使用してください。

画面アイテム「文字列」の場合、関数の処理は以下のとおりです。

- プロパティの画面による各データ変更時の値設定処理（component.set)
- 指定ロケールのデータ設定処理

input (html/js)

ファイル名	ディレクトリ	概要
input (.html/.js)	forma/designer/types/画面アイテムタイプ	「フォーム・デザイナ」画面で編集されたフォームを使用して、ユーザがデータを入力する画面に表示される部品を記述します。 画面アイテムプロパティを設定した時、表示タイプを『入力可』に設定した画面で表示されます。

- input.js
  init 関数の引数に、データ構造とフォーム全体の登録済みの値がプロパティとして与えられます。
  登録済みデータを取得するには、data プロパティからフィールド識別ID と一致するプロパティの値を参照します。
  - 使用例

```
init( arg ){
  var setting = arg.setting; //画面アイテムデータ構造
  var data = arg.data; // フォーム全体の入力値

  var value = data[ setting.input_list[0].input_id ];
}
```

  - 画面アイテム「文字列」の場合

```
var $properties;
var $data;

function init(args){
 var component = args.setting;
 var data = args.data;
 $properties = {
    label : component.item_properties.labels[0][ Module.client.get( 'locale' ) ],
    label_size : component.item_properties.label_size[0],
    initial_value : component.input_list[0].input_properties.initial_value[ Module.client.get( 'locale' ) ],
    input_id : component.input_list[0].input_id,
    input_field_size: component.input_list[0].input_properties.input_field_size,
    max_length : component.input_list[0].input_properties.max_length,
    tab_index : component.input_list[0].input_properties.tab_index
};

 $data = {
    value : $properties.initial_value
 };

 if( typeof data[ $properties.input_id ] != 'undefined' ){
    $data.value = data[ $properties.input_id ];
 }
}
```

- input.html
  ユーザが入力するための入力フィールドと、入力チェックのためのエラー表示関数と、エラー表示クリア関数を記述します。
  エラーハンドリングとエラー表示クリアは以下の形式で記述します。

```
if( !window.formaItems ) window.formaItems = {};

if( !window.formaItems.xxx ){ // xxx = %アイテムタイプ%
   window.formaItems.xxx = { // xxx = %アイテムタイプ%
       AcceptError : function(){
        // エラー表示処理
       } ,
       ClearError : function(){
        // エラー表示クリア処理
       }
   }
 }
```

  - 画面アイテム「文字列」の場合

```
if( !window.formaItems ) window.formaItems = {};

      window.formaItems.product_72_textbox = {
        /** エラー表示 */
        AcceptError : function( event ){
          if($('input[name="' + event.inputId + '"]').parent().find( 'img[src$="exclamation-red.png"]' ).size() == 0 ){
              $('input[name="' + event.inputId + '"]').addClass( 'imfr_item_input_error' )
               .parent().append( '<img src="images/icons/16x16/fugue-icons/shadow/exclamation-red.png" border="0"
/>' );
          }
        },
        /** エラー表示クリア */
        ClearError : function( event ){
          $('input[name="' + event.inputId + '"]').removeClass( 'imfr_item_input_error' )
          .parent().find( 'img[src$="exclamation-red.png"]' ).remove();
        }
      };
  }
```

reference (html/js)

ファイル名	ディレクトリ	概要
reference (.html/.js)	forma/designer/types/画面アイテムタイプ	「フォーム・デザイナ」画面で編集されたフォームを使用して、ユーザが登録したデータを参照する画面に表示される部品を記述します。 画面アイテムプロパティを設定した時、表示タイプを『参照』または、『表示』に設定した画面で表示されます。

- reference.js
  init 関数の引数に、データ構造とフォーム全体の登録済みの値がプロパティとして与えられます。
  登録済みデータを取得するには、data プロパティからフィールド識別ID と一致するプロパティの値を参照します。
  - 使用例

```
init( arg ){
  var setting = arg.setting; //画面アイテムデータ構造
  var data = arg.data; // フォーム全体の入力値

  var value = data[ setting.input_list[0].input_id ];
}
```

  - 画面アイテム「文字列」の場合

```
var $properties;
var $data;

function init(args){
  var component = args.setting;
  var data = args.data;

  $properties = {
      label : component.item_properties.labels[0][ Module.client.get( 'locale' ) ],
      label_size : component.item_properties.label_size[0],
      initial_value : component.input_list[0].input_properties.initial_value[ Module.client.get( 'locale' ) ],
      input_id : component.input_list[0].input_id,
      input_field_size: component.input_list[0].input_properties.input_field_size
  };

  $data = {
      value : $properties.initial_value
  };

  if( typeof data[ $properties.input_id ] != 'undefined' ){
        $data.value = data[ $properties.input_id ];
  }
}
```

- reference.html
  ユーザが入力したデータを表示するための項目、及び、他画面アイテムから値を利用するための、非表示項目を記述します。

クライアントサイド・ファイル

%画面アイテムタイプ% (js)

ファイル名	ディレクトリ	概要
%画面アイテムタイプ%.js	<%WEB_PATH%>/forma/csjs/types	画面アイテムの個別データ構造オブジェクト生成時の初期化関数などの定義、プロパティでの入力チェックエラー発生時のエラーハンドリングを記述します。

- formaItems.xxx 関数
  画面アイテムの個別データ構造です。

アイテムデータ構造のうち、共通なものは別ファイルで定義されているので、ここでは画面アイテムで個別な部分を定義します。
ここで定義された値を優先する形で、基本構造とマージされますので、画面アイテムデータの初期値なども設定しておきます。
xxx の部分は、画面アイテムタイプを使用して下さい。

- formaItems.addPropertiesValidation 関数
  プロパティ画面の入力でエラーが発生した場合の処理を定義します。
  プロパティ項目ごとの個別処理のほか、共通な処理の場合は、imfr_item_prop_tabs のid 属性値を持つdivに対して、changeErrorStyle 関数を実行させれば、itemObject.js ファイルで定義されているとおりに、他画面アイテムと共通なエラー表示処理が行われます。
- formaItems. getFieldList 関数
  itemObject のinput_list に保持する入力項目の中からフィールド一覧に表示対象となる入力項目リストを返却します。
  返却入力項目リストがinput_listの格納順と異なる場合は、fieldListIndexプロパティにinput_listでの格納順を設定する必要があります。
  例えば、input_listに3つの入力フィールド情報を格納していて、フィールド一覧の表示対象にするのは2番目の入力フィールドのみという場合は、2番目の入力フィールドのfieldListIndexに2と設定します。
  格納順が変わらない場合は、指定の必要はありません。

  getFieldList (input_list)

  input_list　itemObject のinput_list

  itemObject のinput_list に保持する入力項目の中からフィールド一覧に表示対象となる入力項目リストを返却します。
  - 実装例

```
formaItems.product_72_departmentSelect.getFieldList = function( input_list ) {
    var inputList = [];
    inputList.push( $.extend( true, {fieldListIndex : 2}, input_list[2] ) );
    return inputList;
  }
```

- formaItems. isTabIndexItem 関数
  実装することで、入力項目のある画面アイテム以外でもタブインデックスを設定できるようになります。
  フィールド一覧へ表示され、タブインデックスを設定することができるようになります。

  isTabIndexItem ( ):boolean

  Returns

  boolean　trueを返却することでフィールド一覧に表示されます。

  - 実装例

```
formaItems.product_80_eventButton.isTabIndexItem = function() {
    return true;
}
```

- formaItems. isForbiddenTabTransition 関数
  実装することで、入力項目のある画面アイテムへのタブインデックスの設定を制限することができるようになります。
  フィールド一覧上での入力項目のある画面アイテムへのタブインデックスの設定を制限することができるようになります。

  isForbiddenTabTransition ( ):boolean

  Returns

  boolean　trueを返却することでタブインデックスの設定を制限することができます。

  - 実装例

```
formaItems.product_72_departmentSelect.getFieldList = function( input_list ) {
    var inputList = [];
    inputList.push( $.extend( true, {fieldListIndex : 2}, input_list[2] ) );
    return inputList;
  }
```

- 画面アイテム「文字列」の%画面アイテムタイプ%.js（textbox.js）

```
(function($){

  formaItems.product_72_textbox = {
    item_id : '',
    item_type : 'product_72_textbox',
    item_exist_dbinput : true,
    item_view_type : {
      REGISTRATION : '0',
      EDIT : '0',
      REFERENCE : '0'
    },
    style : {
      top : 0,
      left : 0,
      width : 300,
      height : 25,
      zindex : 0
    },
    item_properties : {
      labels : [{}],
      label_size : [ 100 ],
      label_styles : [{
        label_background_color : '',
        label_font_color : '',
        label_font_size : 13,
        label_font_family : {},
        label_font_bold : false,
        label_font_italic : false,
        label_font_underline : false
      }]
    },
    input_list : [{
      input_id : '',
      input_data_type : '',
      input_view_names : {},
      input_dbinput : true,
      input_properties : {
        tab_index : 0,
        min_length : 0,
        max_length : 500,
        input_field_size : 150,
        initial_value : {},
        required : false,
        only_ascii : false,
        input_background_color : '',
        border_style : '0',
        border_shadow : true,
        no_frames : false,
        bottom_only : false,
        border_color : '',
        input_font_color : '',
        input_font_size : 13,
        input_font_family : {},
        input_font_bold : false,
        input_font_italic : false,
        input_font_underline : false,
        custom_input_chk : false,
        custom_chk_format : '',
        custom_err_msg : {}
      }
    }]
  };
```

```
    formaItems.addItemDef('product_72_textbox', formaItems.product_72_textbox, function( model, locales ){

        $.extend(true,this,model);
        this.item_id = formaItems.getRandomString(12);

        return this;
    });

    formaItems.addPropertiesValidation('product_72_textbox', function( result ){

        $( 'div#imfr_item_prop_tabs' ).changeErrorStyle( result.target );
    });
})(jQuery);
```

画面アイテムプロパティ共通タグライブラリ

画面アイテムプロパティ共通の入力フィールドには以下の表示用タグライブラリが用意されています。

- *表示タイプ（imart type='formaPropViewType'）*
- *アイテムサイズ・配置（imart type=' formaPropPlacement'）*
- *入力チェック（imart type=' formaInputValidation'）*

また、画面アイテムプロパティ画面を構築する上で以下の便利なタグライブラリが用意されています。

- *連結リストボックス（imart type=' formaDisplayItemSelector'）*

表示タイプ（imart type='formaPropViewType'）

アイテムプロパティ「表示タイプ」の入力フィールドを表示する部分に、下記のタグライブラリを記述します。

```
<imart type='formaPropViewType' item_view_type=XXXX/>
```

item_view_type 属性にはLIBRARY.itemViewType で定義された「表示タイプ」の種別を指定します。

表示タイプの種別	説明	例
LIBRARY.itemViewType.VI_VR_IV	入力系アイテム（表示[input]、表示[reference]、非表示を選択）	文字列、数値、日付など
LIBRARY.itemViewType.VR_IV	表示系アイテム (表示[reference]・非表示を選択)	見出し、各種ボタン、横線など
LIBRARY.itemViewType.VI_IV	特殊な入力系アイテム(表示[input]・非表示を選択)	隠しパラメータ

- LIBRARY.itemViewType. VI_VR_IV を指定した場合の表示例

- LIBRARY.itemViewType.VR_IV またはLIBRARY.itemViewType.VI_IV を指定した場合の表示例

- 実装例
property.js

```
var $propData = {};
function init(arg){
  .....
 if ( typeof LIBRARY === 'undefined') {
    load("forma/designer/types/items_library");
 }
 $propData.itemViewType = LIBRARY.itemViewType.VI_VR_IV;
}
```

property.html

```
<imart type="formaPropViewType" item_view_type=$propData.itemViewType/>
```

アイテムサイズ・配置（imart type=’ formaPropPlacement’）

画面アイテムプロパティ「アイテムサイズ・配置」の入力フィールドを表示する部分に、下記のタグライブラリを記述します。

```
<imart type='formaPropPlacement'/>
```

- 表示例

入力チェック（imart type=’ formaInputValidation’）

画面アイテムプロパティ「入力チェック」の入力フィールドを表示する部分に、下記のタグライブラリを記述します。

```
<imart type=' formaInputValidation' inputListIdx= XXXX validators=XXXX/>
```

- 表示例

inputListIdx 属性には、入力フィールドリストinput_list 配列の何番目かを指定します。
validators 属性には、properties.js で必要なチェック項目をitems_library.js から取得して渡します。

- 実装例
  property.js

```
if ( typeof LIBRARY === 'undefined') {
   load("forma/designer/types/items_library");
}
$propData.inputValidateArgs =
[
   LIBRARY.inputValidator.required, // 必須チェック
   LIBRARY.inputValidator.onlyAscii, // 英数字のみ
   LIBRARY.inputValidator.minLength, // 最小入力文字数（文字列の場合）、最小入力値（数値の場合）
   LIBRARY.inputValidator.maxLength, // 最大入力文字数（文字列の場合）、最大入力値（数値の場合）
   LIBRARY.inputValidator.permitMinus, // 負数入力許可
   LIBRARY.inputValidator.permitDecimal,// 小数入力許可
   LIBRARY.inputValidator.decimalSize // 小数部最大入力桁数
];
```

items_library.js には、下記のように記述されています。

items_library.js

```
inputValidator
[
    required : {key: FRItems.PROP_REQUIRED},
    onlyAscii : {key: FRItems.PROP_ONLY_ASCII} ,
    maxLength: {key: FRItems.PROP_MAX_LENGTH},
    minLength: {key: FRItems.PROP_MIN_LENGTH},
    permitMinus: {key: FRItems.PROP_PERMIT_MINUS},
    permitDecimal: {key: FRItems.PROP_PERMIT_DECIMAL},
    decimalSize: {key: FRItems.PROP_DECIMAL_SIZE}
];
```

タグの内容は下記のファイルに記述されています。

<%WAR%>/WEB-INF/jssp/product/src/forma/designer/types/common/parts/input_validation.js/html

画面アイテム配置時の入力チェック項目の初期値の設定はクライアントサイドファイルの%画面アイテムタイプ% (js)から取得します。

- 関数アイテムの実装例
  func.js

```
input_properties : {
...
required : false,
min_length : '',
max_length : 9999999999,
only_ascii : false,
permit_minus : false,
permit_decimal : false,
decimal_size : 0,
...
}
```

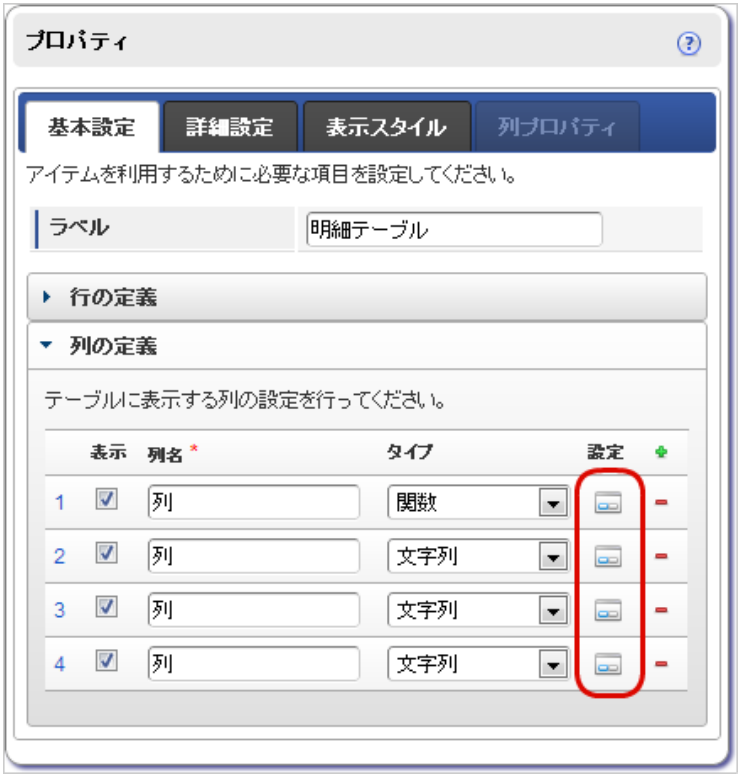
複数のデータ型や、複数の列タイプに対応している画面アイテムの場合、データ型や列タイプが変更された時の入力チェック部品の項目値を初期化するため、各画面アイテムのproperties.html でデータ型や列タイプの変更時に共通の関数”changeDatatype(index)” を呼びます。

- 画面アイテム「関数」のプロパティ画面でデータ型が変更されるタイミングの例

- 画面アイテム「明細テーブル」のプロパティ画面で「明細テーブル」の列タイプが変更されるタイミングの例

また、入力チェック部品の項目を変更する処理を.changeInput(index , value)関数を呼び出して行います。

- 画面アイテム「明細テーブル」のプロパティ画面で表示内容が変更されるタイミングの例（対象列の設定ボタンクリック時）



validators 属性で指定した項目の入力チェック処理は、type.js で実装する必要があります。

- 最小入力文字数と最大入力文字数の入力チェックを使用する場合

```
/** 最小入力文字数 */
result = LIBRARY.validation.properties.valueLength( setting, serverFlg, 'min_length' );
if( result.error ){
    if( serverFlg ){
        return result;
    }else{
        errorArray.push( result );
    }
}

/** 最大入力文字数 */
result = LIBRARY.validation.properties.valueLength( setting, serverFlg, 'max_length' );
if( result.error ){
    if( serverFlg ){
        return result;
    }else{
        errorArray.push( result );
    }
}

/** 最小・最大文字数の整合性チェック */
result = LIBRARY.validation.properties.lengthIntegrity( setting, serverFlg );
if( result.error ){
    if( serverFlg ){
        return result;
    }else{
        errorArray.push( result );
    }
}
```

以上が入力チェック部品の実装方法です。

入力チェック部品で使用される主な関数は下記になります。

- FormaItemObject.changeDatatype(index)
  入力チェック部品の項目の値を初期化する関数です。
  データ型や列の画面アイテム「明細テーブル」で列の入力タイプが変更されるタイミングで利用します。
  引数のindex は、対象オブジェクトのinput_list 配列の何番目かを指定します。
  changeDatatype 関数を利用すると、値を必ず初期化するため、関数を記述する位置に注意が必要です。
  itemObject のinput_list に保持する入力項目の中からフィールド一覧に表示対象となる入力項目リストを返却します。
  - 画面アイテム「関数」のproperties.html でchangeDatatype 関数を利用する場合の実装例

```
expressionDataType.change(function(){
  var oldDataType = component.input_list[0].input_data_type;
  var newDataType = $(this).val();
  …;
  if(oldDataType != newDataType){
      component.changeDatatype(0);
  }
        …
}).val(component.input_list[0].input_data_type).change();
```

- FormaItemObject.changeInput(index , value)
  入力チェック部品の項目を変更する関数です。
  後述のaddInputListener で登録されたリスナー関数を順次呼び出します。
  各画面アイテム特有の処理も同時に実行可能です。
  第一引数の index は、対象オブジェクトのinput_list 配列の何番目かを指定します。
  第二引数のvalue を指定した場合は、addInputListener に追加したリスナー関数の呼び出し時に第二引数として渡されるので、画面アイテムが独自の処理で扱う引数として使用できます。
- FormaItemObject.addInputListener(listener)
  changeInput メソッドが呼び出されたときに実行されるリスナー関数を設定します。
  登録した関数の呼び出し時には、第一引数にchangeInput メソッド呼び出し時に指定したindex の入力フィールドオブジェクトが渡されます。

changeInput メソッド呼び出し時にvalue を指定した場合には、第二引数にvalue が渡されます。
入力チェック部品を組み込むと入力チェック項目と、その値を画面に反映させるリスナー関数が自動的に登録されます。
入力チェック項目ではない項目の表示の切り替え等、画面アイテム特有の処理を行いたい場合にはリスナー関数を追加することが可能です。
入力チェック部品のリスナーの内容は、下記に記述されています。
<%WAR%>/WEB-INF/jssp/product/src/forma/designer/types/common/parts/input_validation.html

連結リストボックス（imart type=’ formaDisplayItemSelector’）

下記のタグライブラリを利用することで、連結した2 つのリストボックスを画面アイテムプロパティに表示させることができます。

```
<imart type="formaDisplayItemSelector"
  displayListId=XXXX undisplayListId=XXXX
  displayListName=XXXX
  undisplayListName= XXXX />
```

displayListId 属性、undisplayListId 属性にはそれぞれリストボックスの上部に表示させるタイトル用のメッセージキーを指定します。
指定しない場合は、以下のデフォルトのメッセージキーが設定されます。

- displayListId 属性 : imfr.item.common.prop.displayItems
- undisplayListId 属性 : imfr.item.common.prop.undisplayItems

displayListName 属性、undisplayListName 属性にはそれぞれ、生成されるリストボックスのname 属性に設定したい文字列を指定します。
指定しない場合は以下のデフォルト値が設定されます。

- displayListName 属性 : displayList
- undisplayListName 属性 : undisplayList

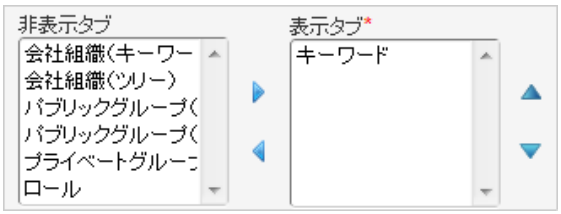
また、リストボックスの内容についてはproperties.html のJavascript で適宜option タグを挿入してください。
option タグを挿入する処理については、このタグを利用した画面アイテム「ユーザ選択」のソースを参考にしてください。

- 実装例

  property.html

  ```
  <imart type="formaDisplayItemSelector"
    displayListId="imfr.item.userselect.label_page.displayItems"
  ```

- 表示例



フォーム･データの保存先

画面アイテムのプロパティなどを格納するフォーム･データは、JSON ファイルとしてStorage のディレクトリ上に保存されます。
JSON とは、JavaScript におけるオブジェクトの表記法をベースとしたデータ記述言語です。
保存ディレクトリは以下です。

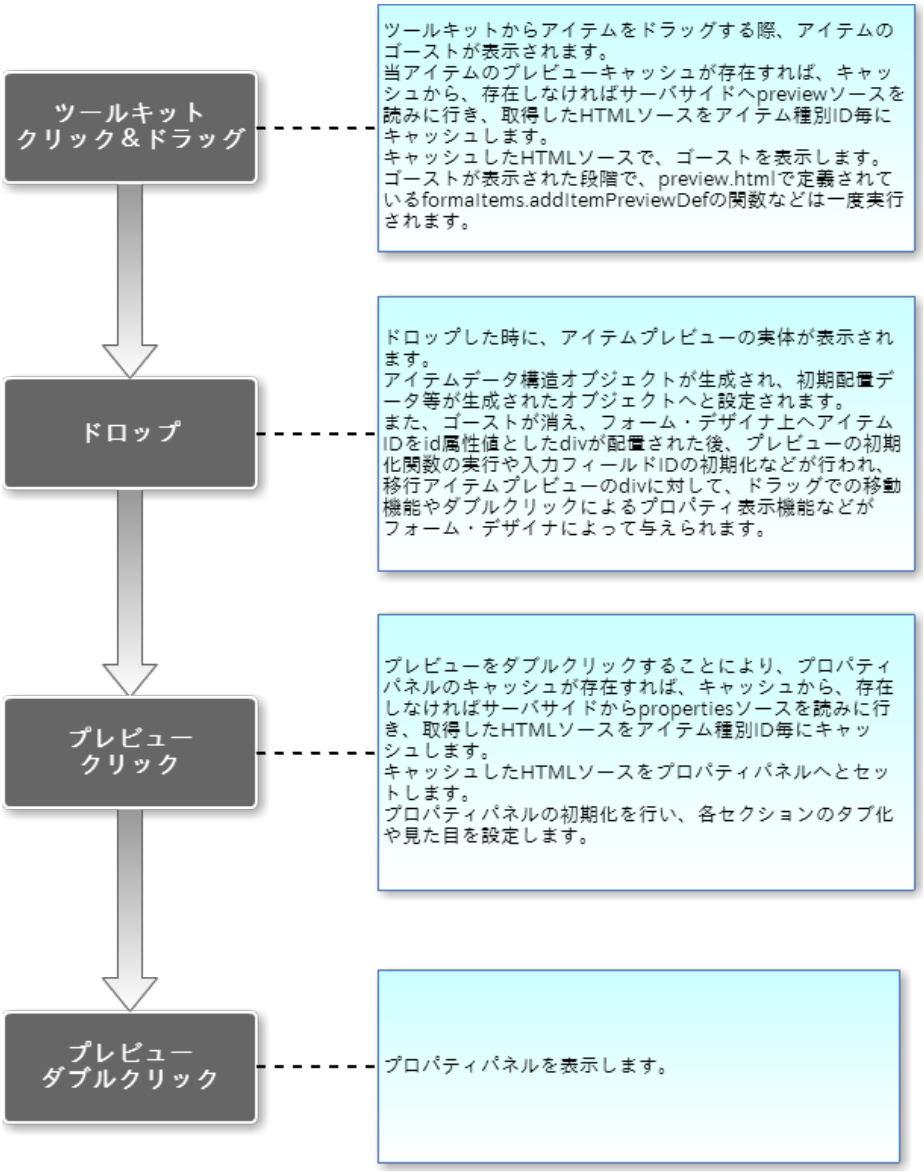
```
<%STORAGE_PATH%>/forma/form/<%アプリケーションID%>/<%フォームID%>/<%フォームID%>.json
```

注意

フォーム設定の整合性を保つため、フォームの JSON ファイルは直接編集しないで下さい。
直接編集した場合、該当ファイルのフォームは正常に動作しなくなる可能性があります。
また、「フォーム･デザイナ」画面での再編集もできなくなる可能性があります。

フォーム･デザイナ画面の表示実行シーケンス

「フォーム･デザイナ」画面において、ツールキットから任意の画面アイテムを配置し、プロパティ画面が表示されるまでの実行シーケンスは以下の通りです。



旧バージョンで作成した画面アイテムの実行

モジュール化

旧バージョンで作成した画面アイテムの資材をモジュール化し、warに組込みます。
詳しくは、「*開発環境のセットアップ*」をご覧ください。

intra-mart Accel Platform対応

旧バージョンで作成した資材については、intra-mart Accel Platform対応を行う必要があります。

- 1. 画面(html)に対しては、『 intra-mart Accel Platform 移行ガイド』の『intra-mart Accel Platform対応』を行ってください。
- 2. ロジック(js)に対しては、API対応を行ってください。
- 旧バージョンの互換APIについてはは「 互換対応表 」を参照してください。

> **注意**
>
> 厳密にintra-mart Accel Platform対応を行う場合は、intra-mart Accel Platformの各プログラミングガイドに従って修正してください。
> また、ルーティング、認可については、各マニュアルを参照してください。

> **注意**
>
> サポートするファイルのエンコードは、UTF-8 のみです。移行前に既存のファイルエンコードを変更してください。

設定の移行

画面アイテムに関する設定方法が旧バージョンから変更となっております。forma-config.xmlに設定されていた内容を移行してください。
詳しくは、「*実装した画面アイテムをシステムに登録する*」をご覧ください。

画面アイテムを「タブ切替」を設定したフォームで利用する場合の注意点

document.ready内で使用する関数は、document.readyより前に定義する必要があります。

「タブ切替」を設定したフォームは、Ajaxによるページ読み込みとなるため、javascriptの読み込みが完了するより速くdocument.readyが実行される場合があります。
そのため、document.readyの処理内で、document.readyの記述より後に定義した関数を使用すると、未定義エラーになります。